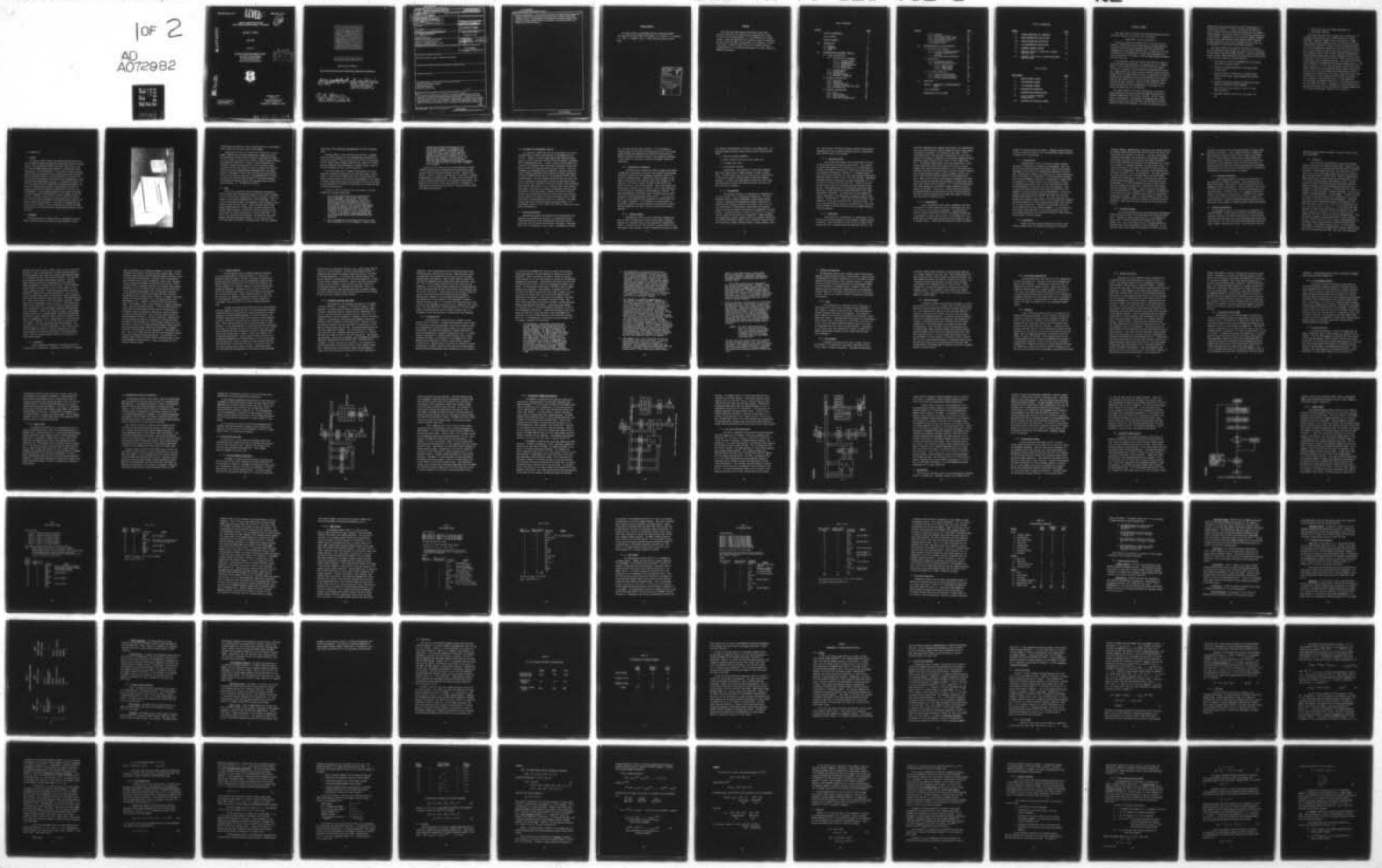
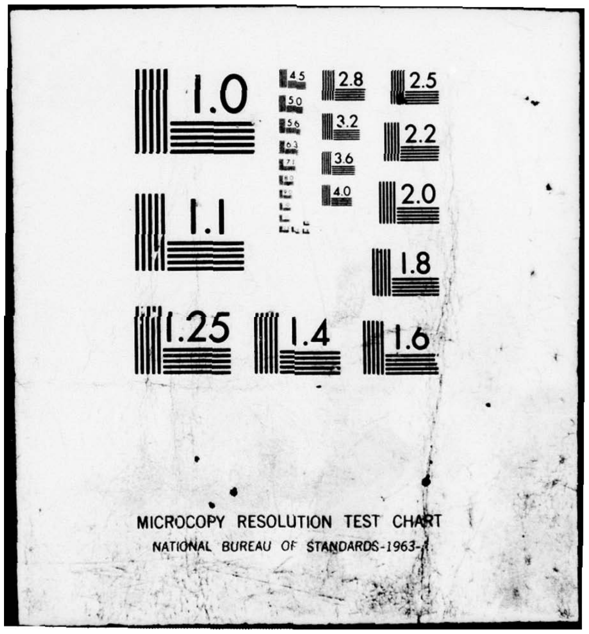
AD-A072 982 MITRE CORP BEDFORD MASS F/G 9/2
ERROR CORRECTION CODING WITH NMOS MICROPROCESSORS: CONCEPTS. VO--ETC(U)
MAY 79 E N SKOOG F19628-78-C-0001
UNCLASSIFIED MTR-3618-VOL-1 ESD-TR-79-125-VOL-1 NL

1 OF 2
AD
A072982




1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ESD-TR-79-125, Vol. I

LEVEL II

MTR-3618, Vol. I

8.S. 12

ERROR CORRECTION CODING WITH NMOS MICROPROCESSORS: CONCEPTS

BY ERIC N. SKOOG

MAY 1979

Prepared for

DEPUTY FOR DEVELOPMENT PLANS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts

Project No. 7010
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-78-C-0001

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

Paul H. Wendzikowski

PAUL H. WENDZIKOWSKI, Capt, USAF
Project Officer

[illegible]

WILLIAM M. SMITH, Jr., Col, USAF
Director, Technological and Functional Area Planning
Deputy for Development Plans

[illegible]

ERNEST L. HATCHELL JR., Colonel, USAF
Assistant Deputy for Development Plans

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 **REPORT DOCUMENTATION PAGE**		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 18 ESD-TR-79-125, Vol-1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 ERROR CORRECTION CODING WITH NMOS MICROPROCESSORS: CONCEPTS, Volume I,		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER 14 MTR-3618, Vol-1
7. AUTHOR(s) E. N. Skoog 10 Eric N. Skoog		8. CONTRACT OR GRANT NUMBER(s) 15 F19628-78-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation P.O. Box 208 Bedford, MA 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 7010
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Development Plans Electronic Systems Division, AFSC Hanscom AFB, MA 01731		12. REPORT DATE 11 MAY 1979 12 105p
		13. NUMBER OF PAGES 103
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 16 7010		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) ERROR CORRECTION CODING MICROPROCESSORS REED-SOLOMON CODE		

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

C3

Error correction coding can be employed in the design of C^3 systems to increase transmission reliability and enhance system effectiveness. The maturation of LSI technology and particularly the ubiquitous microprocessor now allow low-cost simple implementations of error coding hardware. Theoretical fundamentals of linear block codes are reviewed with respect to their suitability to decoder design utilizing 8-bit

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

235 050

JOB

NMOS microprocessors. Key architectural features of NMOS microprocessors are analyzed to determine those characteristics that are best suited to performing decoding functions. A methodology for evaluating MIL-qualified candidate microprocessors is presented that leads to the selection of a microprocessor for the design of the decoder hardware.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the many contributions made during discussions with MITRE staff members D. O. Carhoun, R. D. Haggarty, E. L. Key, S. J. Meehan, and E. A. Palo during the course of this work.

FOREWORD

This report has been prepared under MITRE Project 7010, Low Cost Electronics, for the Electronic Systems Division (ESD) of the Air Force Systems Command. It is presented in two volumes. Volume I treats the theoretical and pragmatic concepts of employing MIL-Qualified 8-bit NMOS microprocessors in the decoding of linear block codes. Volume II details the specific hardware/software implementation of a (7,3) Reed-Solomon decoder utilizing a Motorola MC6800 microprocessor.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

EXECUTIVE SUMMARY

This report covers a portion of the study and hardware activities which were conducted in the fiscal year 77T-78 under,

Project 7010: Low Cost Electronics.

The overall objective of Project 7010 is to provide ESD with technical information and data through analysis and experimentation in specific signal and data processing areas that provide major advantages over technologies now being used. A central part of these investigations is the application of Large Scale Integrated (LSI) circuit technology to C^3 systems in order to impact system performance and life cycle costs.

It has long been a premise of Project 7010 that error coding is one system application area where LSI technology can have a dramatic impact. Earlier project work has shown that linear block codes, particularly a subclass of the Bose-Chaudhuri-Hocquenhem (BCH) codes, viz., Reed-Solomon (R-S) codes, can be an effective signal processing design resource to combat jamming and improve communication reliability. Heretofore, the implementation of such sophisticated error correcting codes has been primarily restricted by the high processing costs associated with their decoding. Project 7010 has identified several viable alternatives for the implementation of such codes, all involving the application of LSI technology.

This report details one of those alternatives: the investigations made into utilizing low-cost 8-bit NMOS microprocessors to implement communication error control functions. Initial investigations (See Appendix) concentrated on fundamental block coding theory including finite field arithmetic and the primary functions required to effect block decoding. For short R-S codes, the use of a

complete BCH algebraic decoding algorithmic[5] implementation is a "computational over-kill". Indeed, this analysis concluded that all but the most simple block decoding approaches were too computationally complex for real-time execution on currently available 8-bit microprocessors. Consequently, it was decided to attempt implementation of a very simple Minimum Distance Decoding (MDD) algorithm. The MDD algorithm for short binary (extended field) R-S codes can be implemented via table-look-up and compare operations.

Key system, hardware, and software architectural features of 8-bit microprocessors were analyzed with regard to their suitability for efficiently supporting the MDD algorithm. From this requirements analysis several "design-desirables" were identified as being critical to the efficient execution of the MDD algorithm:

a. Word Size (W) $\geq$ 4 bits to accommodate R-S codes defined over GF(16) or smaller fields,

b. Indirect or indexed addressing to allow easy access to code tables,

c. Non-destructive bit testability of a masked accumulator for multi-symbol compare operations and erasure masking,

d. On-chip or direct-page register availability for quick temporary storage of received codewords,

e. Fast clock and minimum number of cycles for arithmetic operations,

f. Efficient interrupt structure for code symbol I/O, and

g. Minimum chip count and voltage requirements for easy system interface.

Restricting the search for a suitable 8-bit microprocessor to only MIL-qualified pieces led to a detailed analysis of three candidates: Motorola's MC6800, Zilog's Z-80 and Intel's 8080A. These three CPU's were compared relative to the application design considerations identified in Section 2.0 and an objectively-based selection methodology was developed for the final selection process. Part of that selection process was the careful evaluation of benchmark programs written for each of three candidate microprocessors. These benchmark programs were written to evaluate the extent of the processor's ability to perform a critical "kernel" decoding routine - a code symbol fetch and compare operation. The results of the 18-key factors comparison indicated that for absolute maximum speed (i.e., decoder throughput) the Zilog Z-80 microprocessor was the best choice. Careful examination of the comparison factors, however, pointed out that the Z-80's faster clock speed and increased table storage requirement were responsible for this result. When these factors were normalized (i.e., fairly compared) with the 6800, the Motorola CPU is actually a more efficient choice. If high speed parts are considered, the MC6800B (2 MHz chip) is conclusively the choice over the Z-80A (4 MHz chip). Consequently, it was decided to proceed with the design, test and evaluation of a Motorola MC6800 based (7,3) Reed-Solomon decoder, (Volume II of this report).

1.0 INTRODUCTION

1.1 Purpose

In the past, some C^3 systems have achieved improved reliability and increased throughput through the incorporation of error correction coding. Such system modifications were often "after-the-fact add-ons" which implemented weak codes capable of effecting only modest system gains. The implementation of strong codes has been primarily restricted by high processing costs associated with their decoding.[1] Recent technology advances in LSI microelectronics and particularly the ubiquitous microprocessor now allow low cost realizations of error correction encoding/decoding circuitry. As indicated in Figure 1-1, many algorithms which ran a decade ago on standard minicomputers can be easily executed today with small microprocessor chip sets on standard cards and even completely implemented via new single chip microcomputers. These hardware elements can be integrally designed into new C^3 systems or retrofitted to existing systems to effect increased transmission reliability and enhance system effectiveness. As a result of this, the designer can now consider error coding as yet another inherent design resource. This paper investigates the application of one product of LSI microelectronics, the low cost NMOS microprocessor, to the realization of simple, short block, error coding functions.

1.2 Background

Low Cost Electronics, (Project 7010), is identifying promising signal and data processing areas where recent advances in device

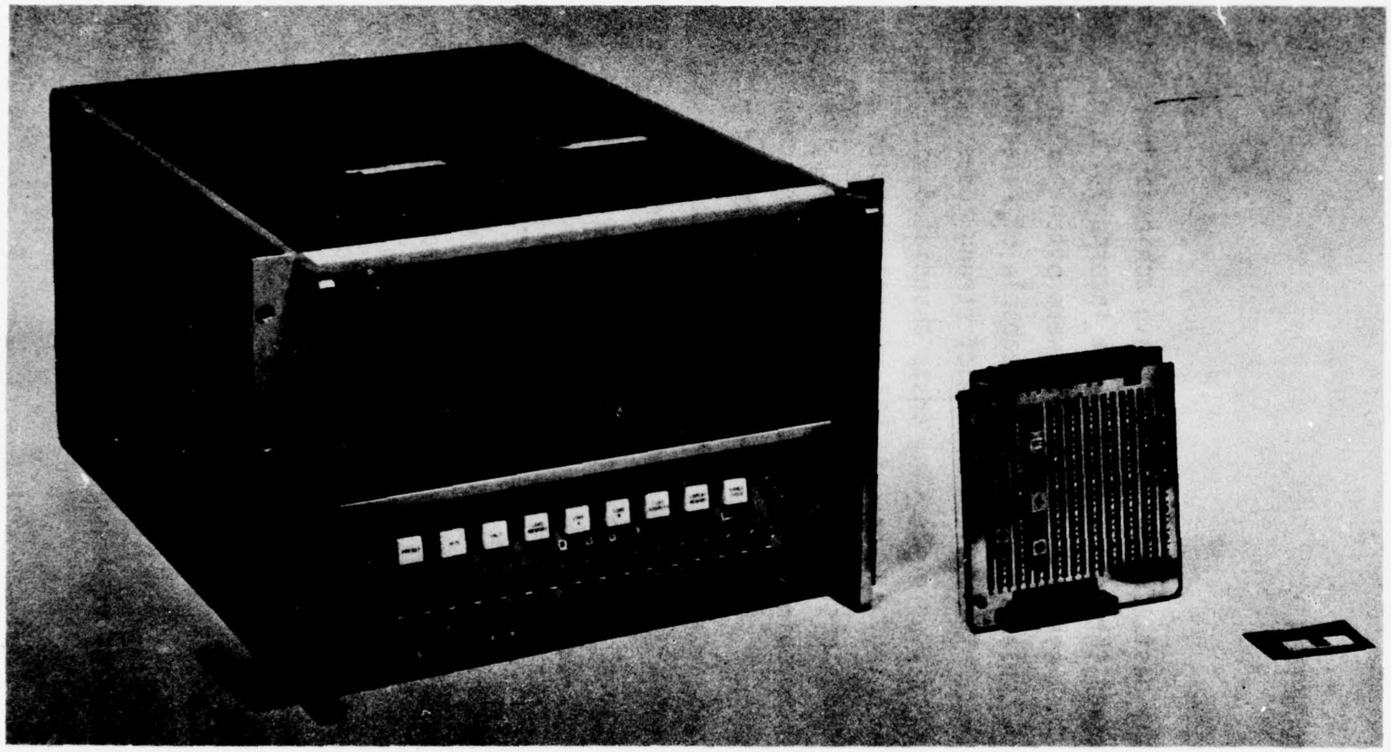


Figure 1-1. Central Processing Unit Comparison

technology can be applied to realize significant gains in performance, and to reduce the life cycle costs of ESD systems.

Recognizing the fact that reliable data transmission is of paramount importance to effective communication, command and control (C^3) operations, Project 7010 has intensely investigated the impact of error correction coding on this problem. Studies already completed have shown that significant increases in system performance are achievable through the incorporation of error correction coding. Investigations have been further narrowed to studies involving cost effective non-binary Bose-Chaudhuri-Hocquenghem (BCH) code implementation techniques.[2] Multi-level CCD structures which perform finite field arithmetic necessary for the implementation of strong non-binary BCH codes have been proposed.[3] This paper documents an alternative technology concept for implementing low-cost error coding hardware, viz., the NMOS microprocessor.

1.3 Scope

Initial error correction coding investigations undertaken as part of Project 7010 concluded that powerful error control strategies can be cost effectively implemented by taking advantage of recent advances in large scale integrated (LSI) circuit technology. The results of those studies state, "The technical issues (involved) can be roughly divided into two classes: those concerned with new technology implementations of strong codes, and those concerned with optimal coding strategies, allocation of design resources, and the relationship of channel characteristics and net coding gain". The latter issue has been quite thoroughly investigated and many fundamental conclusions have been reported. In this paper we shall build upon these conclusions by concentrating on the first technical issue, that of new technology implementations of error correcting codes.

Previous efforts in this direction have resulted in proposed multi-level CCD structures for the implementation of powerful non-binary based codes.[3] While the exploitation of CCD LSI technology in error coding looks extremely promising, this report will investigate the advantages and disadvantages of another prodigy of the recent LSI technology advance - the microprocessor.

It has been shown that C^3 systems' effectiveness can be substantially increased by employing error correction coding. Specifically, a class of cyclic linear block codes, the non-binary BCH codes, have been shown to be particularly advantageous in this regard. In light of these findings,the efforts reported on herein are limited to the implementation of simple non-binary BCH error correcting codes utilizing microprocessors.

In order to limit the scope of this investigation, two basic ground rules have been established:

1. Only low-cost NMOS microprocessor implementations will be considered. This includes only low to medium speed NMOS microprocessors. Since these devices are currently the predominant entry in the microprocessor field, advantage will be taken of their availability, low cost, increasing performance, and design refinements. High-speed bit slice bipolar microprocessor-based designs, while not initially considered in this report, will be investigated as a follow-on to this effort in order to achieve higher data rates, to accommodate increasingly complex decoding algorithms, and to realize improved real-time operations.

2. Direct implementation of non-binary finite field arithmetic operations will not be attempted. Instead, unique

processing algorithms and techniques will be utilized where necessary to accommodate the finite field arithmetic requirements of the selected BCH codes. While it seems completely foreign for a binary-logic microprocessor to perform non-binary field arithmetic, it will be shown that this anomaly is not restrictive. Indeed, by taking advantage of those operations which the binary microprocessor does most efficiently, it will be shown that such "foreign-field" arithmetic can be avoided.

Section 2 of this report analyzes key architectural features of NMOS microprocessors to determine those characteristics which are best suited to performing decoding functions. Based on this analysis Section 3 presents a methodology for selecting a candidate microprocessor for decoder design. Section 4 draws conclusions relative to predicted performance of the candidate microprocessor. The Appendix to this report reviews some theoretical fundamentals of linear block coding which are applicable to low-cost decoder design using NMOS microprocessors.

2.0 MICROPROCESSOR REQUIREMENTS ANALYSIS

It is generally agreed that most any microprocessor can satisfy any application as long as speed and ease of programming are not critical. However, for error coding applications these two factors <u>are</u> very critical since they directly affect the decoder's throughput and the implementation cost. In order to employ low-cost NMOS microprocessors for decoding linear block codes, processors that are best able to perform the decoding operations must be chosen. Given that throughput and programming ease are important, choosing a particular code and decoding algorithm further defines the design requirements. It must be remembered that while all microprocessors exhibit strong and weak points, often the intended application does not require the use of a processor's strong points, nor does the application necessarily suffer from a processor's particular failing. When considering other specific applications, (e.g., different codes and/or decoding algorithms), the analysis of suitable microprocessor features would be conducted in the same manner. However, the results of such an analysis and the conclusions drawn are likely to be quite different. In this study, the microprocessor design requirements must be viewed with regard to the application at hand viz., decoding a (7,3) Reed-Solomon code using a Minimum-Distance Decoding algorithm. The requirements analysis can be divided into software, hardware and system design considerations.

2.1 System Considerations

Although both hardware and software design considerations are important in microprocessor selection, it is generally specific system considerations of the intended application which quickly narrow the field of possible microprocessor candidates. Real-world facts such as availability, price, support, documentation, history, etc., all bear upon the final selection. If it is possible to eliminate many of the potential microprocessors early in the system design consideration phase, then more time will be available for detailed hardware and software analysis of the remaining candidates. In this light, it is important to analyze five general system considerations.

2.1.1 Manufacturer's Commitment

The maturity of a product can often be judged by the amount of commitment a manufacturer has made in the product's establishment. Given that a product has been thoroughly researched (technically as well as marketing-wise), a manufacturer should be eager to show highly visible proof of his complete support and confidence in its viability. The level of support provided can make design life either difficult or easy for the engineer. Due to the complexity of microprocessor devices, each one exhibits certain unique idiosyncracies which may be an advantage or disadvantage to a designer. Adequate documentation, application notes, and design aids can prove invaluable in pointing out these idiosyncrasies. It is very important to firmly comprehend the manufacturer's microprocessor product support intentions. If the designer's selected product is introduced, accepted, and then dropped by the manufacturer, design testing, production, and other life cycle costs may be adversely affected.

2.1.1.1 Software Support

Manufacturers willing to offer significant software support for their microprocessors reflect strong product commitment. Software is a relatively new and alien operation for semiconductor houses. A venture into this area producing a capable family of software support items indicates a sizable investment and should be viewed as a product of determination on the part of the manufacturer. As a minimum, adequate software support should consist of the following items:

1. Cross and resident assemblers,
2. Monitor program (with minimal debug capability),
3. A resident text editor,
4. A non-resident simulator.

Additionally, cross-compilers for high-level languages and a user group library of common programs are useful software tools. The manufacturer may rely on outside sources (contractually or not) such as system "software houses" to make such software available. In either case,representatives should be readily available and willing to answer any software questions and issue software revisions as necessary.

2.1.1.2 Documentation

An excellent indicator of a manufacturer's commitment to his microprocessor is its accompanying documentation. The microprocessor, being a software programmable, highly complex design, requires a completely new documentation approach, radically different than the customary transistor data sheet or the TTL SSI circuit specification. Microprocessor manufacturers are generally equal to the task and supply great amounts of documentation for their products. Documentation must be viewed not so much for its quantity but with regard to its quality and organization. Proper documentation organization requires separate sections on software (instruction set, interrupt execution, support, etc.), electrical characteristics, CPU architecture, support chip descriptions, timing, diagrams, flags etc. Well written, efficiently organized, complete microprocessor documentation can be an effective marketing aid for a manufacturer and a definite asset to the designer.

2.1.1.3 <u>Application Notes</u>

One of the unique aspects of the microprocessor industry is the proliferation of system application notes. This documentation has been recognized by the designer as a principal means of "coming up to speed" on a new device. Manufacturers have realized that impressive applications literature is a formidable weapon in the microprocessor marketplace and can be the deciding microprocessor selection factor for many a timid "first-time" designer. Application information should be regarded with a wary attitude when it is in the form of "see how we did this with this!" Equally as bad are the application notes that illustrate only one inflexible configuration. Often, manufacturers will disseminate application notes which show a unique new design approach using some of their other semiconductor components. Such notes should not be commercially suspect, especially if the manufacturer's pieces are merely illustrated as "representative". Second sourcing often initiates fierce competition in the only unspecified area - application notes. In such a case the designer is obviously the beneficiary of the second source's attempt at gaining market credibility by demonstrating application expertise.

2.1.1.4 <u>Design Aids</u>

Design aids are a manifestation of a manufacturer's commitment to get working hardware into the customer's hands as quickly as possible. These aids generally take the form of special introductory chip sets, prototype boards and supporting card sets. By vertically integrating their manufacturing efforts, many semiconductor houses use this approach to directly compete with small system houses as unbundled microcomputer card suppliers to OEM users. Such an established product line guarantees the manufacturer and his microprocessor some degree of acceptance in the end-user community and more importantly opens a market for his support circuitry (RAMs, ROMs, etc.). The designer should expect the serious manufacturer to offer minimal software aids with his prototype cards such as a monitor programs with some limited debug capability. Well established microprocessor lines offer card sets housed in attractive packages complete with resident software systems including assembler, monitor, text editor, and in-circuit emulation capability for prototyping hardware. These units are called "microprocessor development facilities" and may include such sophisticated peripherals as floppy disks, high speed line printers, and CRT terminals. All of these hardware design aids, when supported by sufficiently capable software,can greatly benefit the designer. Particularly, a sophisticated microprocessor development system can provide detailed design assistance starting from initial timing considerations through final debug of the hardware and software integration effort.

2.1.1.5 Second Source

When a manufacturer enters legal arrangements with other manufacturers to allow second sourcing of a proprietary part, this is an excellent sign of product commitment. The manufacturer's effort to quickly establish his piece as the dominant (volume) part in the market is usually based on faith in his product, and a firm belief that the volume sales realizable are large enough to insure an adequate return. Second sourcing entrenches a product and better insures its continued supply and support. Designers should absolutely insist on recognized second sources of microprocessors before selection of the part for a major design.

2.1.2 Standardization

Standardization is often criticized as being a binding constraint on technological innovation and advancement. While this may be true, there are certainly many design, supply and maintenance advantages to be realized via standardized microprocessors and software. Realistically, however, the concept as far as the microprocessor world is concerned, is a "pipe-dream". With the enormous investments made in the development of complex LSI microprocessors, support chips and software, it is highly unlikely that manufacturers will agree to standardize any product lines. The most the designer can hope to take advantage of is the so called "de facto" standards. These are pieces and designs that have been widely accepted by the user community for their versatility and sound engineering. In the microprocessor field the de facto standards would seem to be the Intel 8080A and AMD's 2900 bit slice family. The Intel product may have achieved this position largely because of the long lead time it enjoyed before the availability of other microprocessors, rather than from any of its unique capabilities. Nonetheless, the designer should not rush to jump on any de facto standard bandwagon. If a microprocessor of different design than that of the de facto standard best suits the intended application, and satisfactorily meets all other selection criteria, then it should be chosen.

2.1.3 Availability

Nowhere else does the word availability connote so many different meanings than in the marketing vernacular of the semiconductor industry. Manufacturers, pressured by the economic realities of a highly competitive market often rush to announce new pieces long before they are "available" for delivery. Full page advertisements, preliminary data sheets, and revolutionary technological advancement announcements characterize this phase of the product development. With initial fabrication and testing off the development line, samples are "available" to "selected customers" for application testing. In reality the product is not readily "available" to everyone and certainly is not in production. The last phase of the development sees the piece "available" in full scale production. This phase continues with announced price cuts as the manufacturer rides the prophetic "learning curve",tweaking his process, increasing yield, and incorporating design enhancements. It is often a long time from piece announcement to full scale development and there are many case histories of products that simply didn't complete the cycle although they were considered "available". The past record of the manufacturer is applicable here. A manufacturer's reputation and credibility in the industry are often intangible factors which are subjectively evaluated by a designer. There is nothing subjective, however, about an excellent production record and a history of successful field applications. These events mark the "availability" the designer must look for.

2.1.4 MIL-Qualifications

One system consideration that eliminates many microprocessors from the selection process is the requirement for military qualified parts. Although the current design trend is to use redundant commercial parts in military qualified environments (e.g., boxes, shelter, etc.), this is not often the case in systems designed for operation under severe environmental or battle conditions. In this selection study of 8-bit NMOS microprocessors our efforts are to be confined to evaluation of the three microprocessors currently undergoing MIL-qualification. Intel's 8080A and Motorola's MC6800 are under test and evaluation by the Air Force's Rome Air Development Center.[13] These processors' slash sheets are being prepared per MIL-M-38510 and will specify circuits to be tested per MIL-STD-883. Additionally, Zilog Corporation offers its Z-80 microprocessor in selected parts which have been screened and tested per MIL-STD-883B. These three pieces will be the only candidates further considered for implementation of the Reed-Solomon (7,3) decoder.

2.1.5 Benchmark Programming

A much maligned method of evaluating microprocessors is the comparison of "benchmark" programs. These comparisons may indeed be biased when a few "typical" routines are selected and programmed on several microprocessors. The bias obviously results from the "typicalness" of the selection, since some microprocessors will outperform others in different tasks. For a specific application, however, this prejudice is exactly what is desired. It is desired to know exactly which microprocessor performs the intended application tasks best. Therefore, benchmark program comparison of key application tasks becomes an extremely important system consideration.

2.2 Software Considerations

The programmer's view of a microprocessor is governed by its software features. Unique software characteristics make a certain processor attractive for different applications by simplifying the software writing and debug task, and by providing more efficient program execution. For the (7,3) Reed-Solomon decoder application this means "tighter" (more efficient) code, faster execution times and therefore increased decoder throughput, and small program storage space requirements.

2.2.1 Word Size

One of the first considerations of any processor-based system is the size of the address and data words. Classically, the address word width is an integer multiple of the data word width, but not always. Microprocessors with small data word sizes, such as 4-bits, must utilize much longer address word sizes to directly access reasonable amounts of memory space. Generally speaking, most NMOS microprocessors available today have data word widths of 4, 8 or 16 bits and exhibit address word widths of 16 bits. The MDD algorithm for small block codes does not require great amounts of memory and therefore the absolute addressing range of the microprocessor is relatively unimportant. The data word size, however, represents a critical parameter in decoder performance. The main data element in the MDD algorithm is the code symbol. As shown in Section A1.1, this finite field element must be represented as an m-tuple over GF(2) in the binary realm of the microprocessor. Therefore, a relationship must be established between m and the width of the data word, which we shall call W. For the (7,3) Reed-Solomon code $m = 3$, and for a (15,k) R-S code $m = 4$. It has already been shown that the full table search MDD algorithm is difficult if not impossible to spatially or temporally realize with longer R-S block lengths and therefore a limit of $m \leq 4$ would seem appropriate. This would seem to suggest a data word width of $W = 4$ bits, sometimes called a "nibble" or half of an 8-bit "byte"! Data manipulation in the decoder would then be equivalent to handling m-tuple binary code symbols. The tradeoff involved with $W = 4$ is the execution time involved in fetching and comparing data (symbols) one-at-a-time. Most 4-bit microprocessors are PMOS, slow, multi-cycle addressing machines which require several address cycles to fetch a particular 4-bit word. Since a large portion of the MDD algorithm requires fetching codewords from a table and comparing them symbol-by-symbol, it is essential that this fetch time be minimized. This objective can be achieved through "multiple-symbol fetching". Multiple-symbol fetching can be best accomplished with the largest word width microprocessor currently available, the W = 16 bit processor. These processors can provide up to four 4-tuple symbols or five 3-tuple symbols per fetch. The problem then revolves around the processor's capability for "nibble manipulation". This is to say, given a multiple-symbol fetch, how many nibble (symbol) operations must be performed to effect symbol comparisons. Microprocessors with $W > 4$ offer only limited (if any) direct nibble operations. Nibble moves must be accomplished via bit-shifting, or more efficiently, fetch and mask operations. Using a non-destructive masking operation, the results of a multiple-symbol comparison (XOR operation) can be recognized. Clearly, for larger and larger W, fewer symbol fetches and word (i.e., multiple-symbol) comparisons need be made. The number of symbol comparison results that must be checked, however, is independent of W and is always equal to N, the block size of the code. Therefore, it is concluded that for decoding (N,k) R-S codes, W = 16 is the optimum data word width parameter. This dictates the use of a 16-bit microprocessor. Lest a final conclusion be reached too quickly, it must be remembered that 16-bit microprocessors are relatively new entries in the microprocessor field. This fact becomes extremely dominant in the analysis of other design factors.

2.2.2 Addressing

Various addressing techniques are provided as part of a microprocessor's architecture. Regardless of the specific techniques used, the objective of all addressing schemes is the same - to allow references to any point in the microprocessor address space. Microprocessors with extremely flexible addressing modes are better suited to a wide range of applications. Generally, most microprocessors offer the most trivial forms of addressing, "direct" and "immediate". Savings in program development and execution time, however, result from the ability to reference locations in the address space without including a full set of address bits in each and every instruction. To this end, many microprocessors offer "abbreviated" addressing utilizing only 8-bits to define the lowest 256 memory locations. An extension of this concept yields "relative", or "paged" addressing schemes utilizing some register (e.g., the Program Counter, Base Page Register, etc.) contents to further define the complete address. This concept, in turn, can be related to "indexed" addressing using a specially designated register. If the CPU architecture provides on-chip registers, these locations may be considered part of the address space and accordingly addressed by so-called "register" or "implied" addressing modes. One of the most sophisticated addressing methods available is "indirect" addressing. Combined with on-chip registers this mode can be enhanced to "register-indirect" and feature "auto-increment" and "auto-decrement" capability as well. The power of indirect and indexed addressing becomes readily apparent when an application program requires the manipulation of members of arrays of varying sizes, or the development of relocatable software. The (7,3) R-S MDD algorithm requires an extreme amount of table look-up and scratchpad storage manipulation. The candidate microprocessor should therefore exhibit powerful addressing modes to ease the programming of such operations, reduce the program size, and increase execution speed.

2.2.3 Internal Registers

On-chip registers are an extremely important characteristic of CPU architecture for three main reasons: 1) they allow additional addressing modes in the CPU, 2) register to register operations generally provide faster data manipulation operations than memory-reference instructions and, 3) if the on-chip registers are flexibly capable, fast interrupt response is possible through "bank switching" techniques. Although as a rule it can be stated "the more on-chip registers the better", the flexibility of such registers must be carefully analyzed. Simple "general purpose" or "scratchpad" registers are adequate for operand storage and retrieval, but are "bottlenecked" if they cannot directly provide arguments to an Arithmetic Logic Unit (ALU) or be used in enhanced addressing modes.

In order to provide maximum flexibility the on-chip registers should be capable of receiving as well as sourcing operands to the ALU. Furthermore, the CPU's instruction set should directly utilize these registers to provide immediate operands or indirect addresses. Additionally, they should facilitate rapid handling of return addresses and status indicators during subroutine or external interrupt operations. One of the most powerful uses of on-chip registers is stack pointing. If the return addresses of subroutines and interrupt handlers are stored in a push-down last-in-first-out (LIFO) stack, such a stack can be realized with on-chip registers or external read/write memory. With on-chip storage there is always a limit to the "depth of subroutine/interrupt nesting". Such a constraint must be carefully regarded when programming such microprocessors, for if an on-chip stack overflow does occur, there is normally no alarm indication and software malfunction (or lost data) will occur. On the other hand, off-chip stack storage offers virtually unlimited subroutine nesting and facilitates the implementation of very powerful stack algorithms.

Utilizing off-chip storage for the stack, only a stack pointer need be stored in an on-chip register. Multiple stack-pointer registers in conjunction with register-to-register operations provide a very powerful stacking capability. It is obvious that maximum register flexibility requires an enormous penalty in CPU design complexity, a tradeoff that quickly becomes prohibitive. However, multiple accumulators, processor status registers, index registers and stack pointers are all desirable features in a flexible CPU architecture. The (7,3) R-S decoder application under consideration here, can benefit greatly from the increased execution speed and data manipulation ease afforded by the use of multi-purpose on-chip registers.

2.2.4 Arithmetic and Logic Unit (ALU)

A microprocessor's ALU represents the heart of its "number-crunching" capability. Although it is the central element in these operations, a sophisticated ALU does not guarantee efficient number crunching. Many other processor features such as addressing, I/O capability, interrupt handling, instruction set, etc., have a far more profound effect on the microprocessor's arithmetic and logic processing capability. Nevertheless, the structure and adequacy of the microprocessor's ALU is important in most applications. Basically, an ALU must perform arithmetic and logic operations on binary data in increments of the basic data word size (W). Since most microprocessors perform two's complement arithmetic, this requires the ALU to perform binary addition, one's complement, Boolean operations (i.e., AND, OR XOR, NOT) and single bit left or right shift operations. More complex data operations, and multi-byte operands, may be supported by the microprocessor's instruction set and control logic. The results of an arithmetic or logical operation are contained in the accumulator of the CPU and reflected in the flag positions of the Processor Status Word. Complete status flag indicators support the development of sophisticated instruction sets and therefore increase computational capability. Most microprocessors offer the basic ALU status flags indicating carry, overflow, zero and sign. Many processors now offer additional flag indications of half-carry, parity, interrupt enable/disable, and subtract. Many of these flags are specifically included to accommodate Binary Coded Decimal (BCD) operations in the ALU. The (7,3) R-S decoder application does not require any BCD capability. The multi-symbol compare operations of the MDD algorithm require the Boolean XOR operation and a check of the accumulator zero (Z) flag for the results. Symbol-masking for multi-symbol comparison results and erasure gating can be performed with the Boolean AND operation. Whether or not the symbol-masking operation is "non-destructive" to the contents of the accumulator in order to position the status flags, is a function of the instruction set, since the ALU is only commanded by the Control Unit's decoding of the instruction. This "non-destructive" masking operation is of critical importance in the MDD algorithm since it affects execution time (i.e., throughput via the time required for register saves and refetches).

2.2.5 Instruction Set

A microprocessor's instruction set is perhaps the most touted of all its features. Although often marketed on sheer numbers of instructions alone, microprocessors that do not include the necessary complementary architecture cannot support very sophisticated instructions. The three CPU architectural features which most impact the microprocessor's potential instruction set are: 1) internal register organization, 2) allowed addressing modes, and 3) status indicators available. As discussed in 2.2.2 through 2.2.4, these features must be optimized to realize a sophisticated, powerful instruction set. Generally most instructions are designed to operate on data words of width W, the basic data width of the microprocessor. Restricting the microprocessors under consideration in this report to

W = 8-bit militarized NMOS CPUs, this basic word is an 8-bit byte. Some instructions will provide operations on double word or 16-bit quantities. These instructions are normally limited to loading a 16-bit register such as an index register or pointer and incrementing/ decrementing that register. Sophisticated instruction sets for 8-bit microprocessors will offer some 16-bit (doubleword) arithmetic operations. For manipulation of BCD digits, half-word, 4-bit or nibble operations are desirable. Unfortunately, at present all of the available 8-bit NMOS microprocessors provide BCD arithmetic via the ALU and special status flags, but do not provide general nibble manipulation instructions. In the case of R-S decoding it has been shown (see 2.2.1) that a 4-bit symbol is optimum for MDD. Were direct nibble manipulation instructions available, the MDD algorithm could be performed more efficiently. Instructions which operate on individual bits of a word find widespread application in control and cryptographic applications. Instructions designed to test, set, reset, and move bits are not available in most 8-bit NMOS microprocessors, nor does the MDD algorithm of this application require their use. A microprocessor's instruction set can be divided into five general categories of instructions:

1. I/O Instructions - Input/output instructions are of prime importance in most microprocessor applications. Efficiently getting data into and out of the microprocessor is of utmost importance in optimizing throughput. Two general I/O schemes are generally recognized: (1) "memory mapped" with memory reference instructions, and (2) special I/O instructions which do not address the microprocessor's main address space. Whichever scheme is utilized it is important not to tie up I/O operations by a CPU architecture which exhibits "accumulator-bound" I/O operations. This is to say all I/O operations must use the accumulator and only the accumulator as the source and destination register. Furthermore, an architecture which facilitates multi-word I/O transfers can be an effective low speed alternative to DMA.

2. Load Instructions - Load and store instructions are designed to efficiently move data into, around, and out of a microprocessor. As with I/O instructions, the CPU architecture must not restrict these load operations to a specific register (e.g., an accumulator). 8-bit and 16-bit loads from main memory to any on-chip register(s), and register-to-register moves must be provided. Load instructions are the chief beneficiaries of the efficiencies provided by sophisticated addressing techniques. For example, load operations involved in array processing and file handling operations particularly benefit from indirect and indexed addressing. As in I/O instructions, the capability to load multiple words (i.e., block moves) can be a very useful feature.

3. ALU Instructions - Arithmetic operations (e.g., AND, SUBTRACT, INCREMENT, DECREMENT, COMPARE, etc.), and Boolean Logic operations (e.g., AND, OR, XOR, etc.) are performed in the Arithmetic Logic Unit via specific ALU instructions. While most microprocessors provide these basic ALU instructions, differences arise concerning where the operands are derived from and where the result can be stored. Once again, "accumulator bound" ALU operations are least desirable. Ideally the CPU's internal registers and external memory should be able to source and sink the operands. Additionally, block operations such as "compare and step" are useful memory search instructions. Individual bit testing, setting and resetting with a single ALU instruction is an extremely desirable feature, particularly for control applications. Special arithmetic instructions which operate on BCD data are often used for efficiently processing instrumentation data without requiring BCD-binary conversion. The MDD algorithm makes maximum use of the XOR operation with indexed or indirect addressing for performing comparisons between the received codeword and valid codewords stored in the codetable.

4. JUMP, BRANCH & CALL - These instructions provide the microprocessor with decision direction. By altering the Program Counter, instruction execution sequences can be varied depending on dynamic run-time conditions. Both conditional and unconditional jumps and branches are extremely effective for transferring program control. Return instructions return program

control to the software point in effect before the Call or Interrupt. Conditional returns are very powerful means of continuing or discontinuing subroutine execution conditioned on some specific run-time criteria.

All of these instructions are enhanced by a wide variety of addressing modes. Branches or jumps that can be relative, paged, indexed, direct, or indirect provide the programmer many efficient ways of directing program flow. The MDD algorithm requires many conditional decisions which may employ these instructions. The trade-off in this regard, particularly with subroutines, is the time required for subroutine linkage to be established and broken down, and its effect on decoder throughput.

5. Status Instructions - We have seen in 2.2.4 that every microprocessor offers some selection of status flag indicators displayed in the Processor Status Word or Flag register. Instructions to directly act on these flags are often important in directing program flow. These instructions normally consist of setting or clearing a particular flag. Many microprocessors' instruction sets do not include these instructions. This is not considered a major deficiency, however, because the flags can be set or reset by performing some register-to-register operation which requires no more execution time than any set/reset flag instruction.

   Example: The Z-80 microprocessor does not provide a reset carry bit instruction. This operation can be simply performed by logically ANDing the accumulator with itself; an operation which executes in the minimum instruction time.

   The most widely used status instruction is normally the clear or enable interrupt flag. Using this instruction the microprocessor can control its interrupt environment. This status instruction is very important in the MDD algorithm for interrupt receipt of code symbols while decoding previously received codewords.

2.3 Hardware Considerations

Microprocessors execute their software, hopefully efficiently, through a unique amalgam of hardware idiosyncracies. When evaluating microprocessors for specific applications it is not enough to merely consider the ramifications of their software features. A designer must also carefully weigh the advantages and disadvantages offered by the microprocessor's hardware design. Hardware in this sense means not only the architectural characteristics of the monolithic CPU chip itself, but the support requirements of the microprocessor as well. There are ten specific hardware considerations that should be considered.

2.3.1 Clock

All microprocessors, regardless of their technology, require some sort of clock signal to synchronize their operation. This, however, is where the similarity ends. Some microprocessors require an external (off-chip) clock generator. This generator may be required to provide the microprocessor with high-level, multi-phase clock signals. Contrasted with this approach are microprocessors which include on-chip clock generation, multi-phase, and driver circuitry. Such chips merely require an external crystal, RC circuit or single-phase low level clock for proper operation. Microprocessors offering on-chip clock generation are easier to design into a system, require fewer packages and avoid complex multiple phase clock adjustments.

2.3.2 Chip Voltages

The number of different power supply voltages that must be provided to a microprocessor chip affects distribution wiring, circuit layout, and power requirements. Many NMOS microprocessors, like their ancestral PMOS forefathers, require multiple power supply voltages, generally ±5 and ±12 volts. When interfacing with bipolar analog circuitry, these voltages are sometimes available, but newer generation devices only require a single +5 volt supply. This is highly desirable when designing a printed circuit card power plane layout, and is compatible with TTL logic circuits. The advantages of a 5-volt-only microprocessor chip may be quickly outweighed, however, if the designer is constrained to use support chips which require multiple voltages (e.g., UV-erasable Read Only Memories or dynamic RAMs).

2.3.3 Power Dissipation

The amount of power a microprocessor dissipates is a hardware characteristic which is an integral factor in the system design. Naively, we could conclude that the smaller the power drain the better. This reasoning would lead to an ultra-low power CMOS microprocessor chip such as the RCA 1802. Obviously, this philosophy neglects the intended application. The application must ultimately drive all microprocessor design considerations, both hardware and software. Some applications, such as satellite service or other systems with critical power budgets, may demand low power CMOS microprocessors. Other real-time, heavily interrupt-driven applications might require processing speeds only available through bipolar (Schottky or ECL) bit-slice architectures with their attendant large power drain. In the R-S decoder application, power is not considered a critical resource, but neither is there a requirement for the all-out processing speed of a high-power bipolar design. By restricting ourselves to the consideration of only 8-bit NMOS microprocessors, the average power dissipation of our CPU is predetermined by available devices.

2.3.4 Logic Family Compatibility

Most microprocessors offer some sort of TTL compatibility at input and output pins. Those that do not must be appended with MOS-TTL level converters. A microprocessor which does offer TTL compatible pins must be looked at closely. Often a part may be capable of sourcing several TTL loads but capable of sinking only 1 TTL load. This common characteristic need not cause any difficulties when interfacing with only MOS chips, but if interface requirements dictate many TTL chips, then buffers will be necessary to increase the fan-out. Since the majority of today's medium-speed logic is TTL based, TTL-capability is a very desirable hardware feature.

2.3.5 Packaging

Logic circuits are widely available in a number of different package styles and types. Certain "standard" packages have evolved as a result of the popularity and acceptance of the RTL/DTL/TTL logic families. Certainly, the most familiar commercial packages are the 14 and 16 pin Dual-In-Line (DIP) plastic types and the military ceramic DIPs and "flat-paks". Extension of fabrication techniques into LSI circuit technology has produced larger packages (e.g., 18, 20, 22, 24, 28, 40, 48 and 64 pins) of the same types. Most 8-bit NMOS microprocessors have settled into 40 pin ceramic or plastic DIP packages. Since automatic insertion and testing equipment has been designed to accommodate this package, it is economically beneficial to stick with this type of packaging. Although plastic commercial packages are readily available and enjoy the lowest cost, ceramic DIPs are best suited for hermeticity and the other military qualification requirements specified by MIL-STD-883.

2.3.6 Interrupt Structure

Interrupts are a very important design consideration in an application where real-time response to external events is required. The interrupt capability of a microprocessor is governed by its hardware structure and its supporting software. Two questions must be asked about the microprocessor when considering interrupt sequencing: 1) What happens to the program being executed at the time of the interrupt, and 2) How can the microprocessor determine which device is interrupting and what priority it has been assigned? The first question is answered by what actions that occur prior to the acknowledgement of an interrupt request. Some provisions must be made to "freeze" the processing status of the program being interrupted. This entails saving the point in the main program (Program Counter) where execution was interrupted, the contents of the accumulator, processor status register, and possibly other important on chip register contents. These save-operations may be performed automatically by the microprocessor upon recognition of an interrupt, or may be included as instructions in a software interrupt routine. Automatic saves realize a savings in interrupt code but cause "generalized time" penalties by executing at every interrupt, whether or not they are necessary. The second question requires indentification of multiple interrupt sources and priority of those interrupts. This answer could be obtained by dedicating many pins of the microprocessor to interrupts; each pin representing an individual interrupting device and assigned priority. Alternatively, one pin can be dedicated to interrupts with external circuitry supplying interrupter identification and priority. The latter scheme requires some intelligent support chips, a technique prevalent in current 8-bit NMOS microprocessors. Some third generation chip families are solving the problem by using "daisy-chain" hardware for identification and priority arbitration. This solution still requires some support circuitry intelligence for daisy-chain enable control. Regardless of how these two interrupt questions are answered, the microprocessor must posses the capability of controlling its interrupt environment. This is most easily accomplished by a software interrupt enable/disable instruction. Most microprocessors offer this "interrupt masking" capability, although some offer an additional "non-maskable" interrupt that will always recognize and set on an external event. Such events as power-loss may require the immediate movement of critical data to nonvolatile memory for retention and can be unconditionally initiated by such an interrupt. The (7,3) R-S decoder application requires an interrupt capability. This interrupt is used to service codewords which are received while decoding is in progress on previously received codewords, and therefore by itself does not require a complex multi-level interrupt structure.

2.3.7 Direct Memory Access (DMA)

DMA represents a very efficient method to effect high-speed data interchange between a microprocessor's main memory and peripheral units. The alternative to DMA requires that some firmware routine read/write each data word between main memory and the peripheral device, an option termed "programmed I/O". DMA requires that the microprocessor not be working in main memory at the time the DMA transfer is occurring. This result can be achieved in either of two ways: 1) Disable the processor during DMA time, or 2) Allow DMA only during those times that the CPU is not using main memory. These approaches, termed Cycle Stealing and Simultaneous DMA, respectively, are both valid methods and the implementation choice between them revolves around the microprocessor's DMA capability, normal processing load, and program timing requirements. The (7,3) R-S decoder application can utilize DMA transfers to load received codewords into main memory simultaneously with CPU decoding operations. Such operations would increase the decoder throughput when contrasted with programmed I/O.

2.3.8 I/O and Memory Expansion

The ability to adapt a microprocessor-based design to provide the additional processing required for possible later design enhancements can be a chief contributor to minimum life-cycle costs. The main concern in this area is the ability to directly expand the accessible main memory and the number of I/O channels. These two expansion possibilities are most influenced by the software provisions of the microprocessor and the available hardware interface. As discussed in 2.2.2 and 2.2.5, addressing range and modes directly affect memory and I/O expansion. Additionally, multiplexed or non-multiplexed microprocessor pin-outs, decoded or non-decoded control signals, and drive capability are the hardware features most involved in electrical expansion design. An application such as the R-S decoder might take full advantage of additional memory and/or I/O ports for further accommodation of longer codes, outboard special purpose hardware, or additional demodulator functions such as estimation/detection processing.

2.3.9 Minimum Chip Count

A one-chip microprocessor is really a misnomer when the number of additional support chips necessary for operation are considered. A one-chip CPU packaged in a large DIP will often provide all the necessary address, data and control lines for easy system interfacing. On the other hand, a one-chip CPU packaged in a relatively small DIP package provides these signals on multiplexed pins and requires external packages for demultiplexing and control. Microprocessors that require external clock generator/drivers, system controllers, or other support chips for even a minimum chip implementation obviously require more space, weight, power, and are inherently less reliable in operation. A true one-chip CPU must include a clock generator and all system control logic to provide all necessary data, address and control lines in a non-multiplexed fashion. The one-chip microprocessor must not be confused with the one-chip microcomputer. The latter is a monolithic device which includes on-chip program storage (ROM), data register storage (RAM), and I/O logic. The monolithic microcomputer is truly the ultimate in minimum chip system design. The R-S decoder, intended as part of a larger system, should take advantage of the benefits realized by minimum chip designs.

2.3.10 Support Chips

The power and flexibility of a microprocessor can be increased by performing complex and/or time consuming processes in external logic. State-of-the-art LSI is now providing microprocessor support chips which, under software control, provide synchronous/asynchronous I/O operation, counter/timer functions, DMA control, floppy disk and magnetic tape control, video generation, A/D/A conversion, etc. Microprocessor selection should include consideration of the family of support chips available for each particular device. Support chips may be required in the original design to implement the intended application or may be considered as increased processing potential for later design enhancements. The use of highly sophisticated support chips to achieve increased system performance is an attractive alternative to increasing a system's clock speed, adding ROM and RAM, or having to redesign with a different microprocessor.

3.0 MICROPROCESSOR SELECTION METHODOLOGY

In the previous section of this report, specific microprocessor design considerations were analyzed with regard to implementing the MDD algorithm. It now becomes necessary to select a microprocessor that will satisfy the design requirements resulting from that analysis. The selection process should be as objective as possible. In this section, an approach to microprocessor selection for the (7,3) Reed-Solomon decoder is presented. Any selection methodology that assigns relative weights to different evaluation factors can be immediately suspect, and therefore the unweighted method employed herein is believed to be a more objective approach.

The subjective pitfalls of figure-of-merit weighting arise from the fact that some applications include "critical factors" (i.e., design constraints) which require emphasizing their relative importance over all other factors under consideration. This is normally accomplished by assigning a relative merit weighting figure to each design factor. In such cases great care must be taken when assigning these "multiplying" figures. Easily biased results can quickly arise from the inclusion of overinflated (multiplied) evaluation factors depending on how one plays the "numbers game".

Relative figures of merit for the design factors of the (7,3) Reed-Solomon decoder application under consideration here have not been used; however, it can be argued that certain design considerations have been implicitly declared critical. Certainly, the objectives of low-cost, medium speed, flexible components dictated NMOS 8-bit microprocessors, and the MIL-qualified requirement immediately eliminated most of the potential candidates. Therefore, these design factors have not been considered further but have affected the microprocessor selection process by allowing closer examination of the three remaining microprocessors.

In the selection methodology presented here, the microprocessors' performance in each design factor category has been rated by strict adherence to a few basic evaluation rules (see 3.3). These rules have been applied in the context of the considerations of the design factors as presented in Section 2.0. This microprocessor selection methodology is applicable to other designs if the evaluation rules are applied in the context of the intended application and can include relative weighting for "critical factors" if necessary.

In this section the three MIL-qualified microprocessors are examined, application benchmark programs written for each microprocessor, and the results of these benchmarks, as well as 17 other design factors, are evaluated and compared.

3.1 MIL-Qualified Candidates

As discussed in 2.1.4, the system consideration factor that eliminated many 8-bit NMOS microprocessor candidates was the availability of MIL-Qualified pieces. Applying the criteria, we are left with three candidate microprocessors: Intel's 8080A, Motorola's MC6800, and Zilog's Z-80.

3.1.1 The Intel 8080A Microprocessor

The 8080A is a monolithic 8-bit parallel processing unit (CPU) utilizing silicon gate, depletion load, NMOS technology.[14] As illustrated in Figure 3-1, the 8080A's architecture includes seven 8-bit programmable registers (including one accumulator), a 16-bit stack pointer, and a 16-bit program counter. Six general purpose registers may be addressed individually or in pairs to provide both

IA-52,860

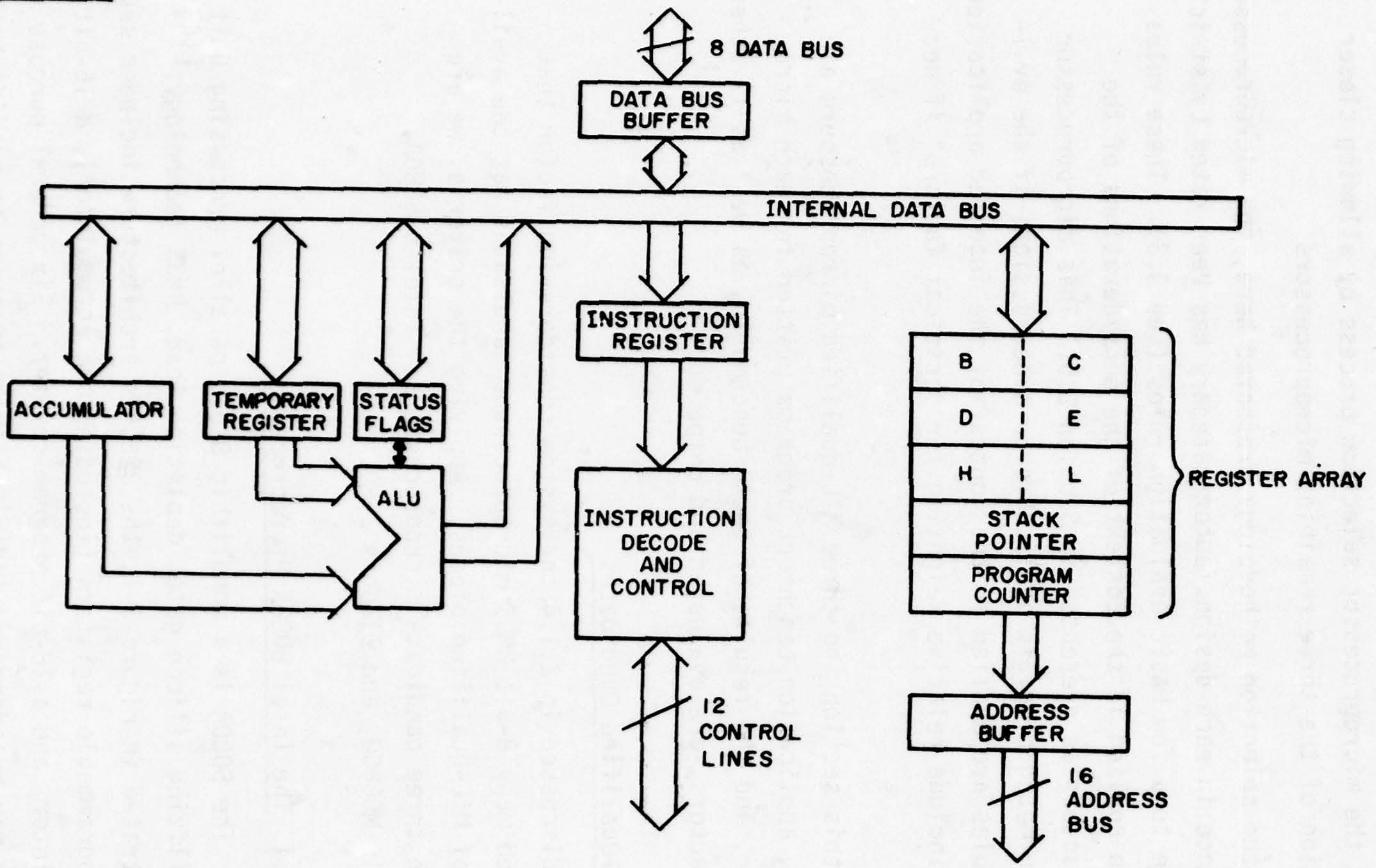


Figure 3-1: 8080A MICROPROCESSOR ARCHITECTURE

single and double precision operators. The stack pointer points to an expandable stack which must be implemented in external RAM. This last-in-first-out stack can be used to store and retrieve the contents of the accumulator, status flags, program counter, or the six general purpose on-chip registers. The sixteen bit stack pointer controls the addressing of the external stack. In this manner, the 8080A can handle virtually unlimited subroutine nesting from multiple-level priority interrupts by rapidly storing and restoring the processor's status. Arithmetic and logic instructions set or reset four testable flags. A fifth flag provides decimal arithmetic operations.

Intel's 8080A was the first of the so-called "second generation" 8-bit microprocessors and is a design enhancement of the old 8008. It is by far the most widely used 8-bit NMOS microprocessor available today. There has been a lot of software written for the 8080A and much design experience gained by its use. Nonetheless, the 8080A does exhibit some glaring deficiencies. For example, it is first and foremost really a 3-chip microprocessor, when considering the necessity of a clock generator chip and a system controller chip. Additionally, its three power supply voltages are often a design nuisance. Software-wise, its lack of indexed addressing can be an impediment to some programming tasks, but its versatile register - indirect addressing capability finds wide application. The 8080A is firmly entrenched in the microprocessor marketplace with strong support, a myriad of second sources, and a vast family of very powerful support chips. Third generation devices, while offering vast performance improvements over the 8080A are exhibiting software compatibility with this ubiquitous device, a strong testimony to its position in the microprocessor field.

3.1.2 The Motorola MC6800 Microprocessor

The MC6800 is a monolithic 8-bit parallel central processing unit (CPU) fabricated in silicon gate, depletion load, NMOS technology.[15] As illustrated in Figure 3-2, the 6800's architecture includes two 8-bit accumulators, a 16-bit stack pointer, 16-bit program counter, a 16-bit index register, and an 8-bit condition code register. The stack pointer is used to reference a stack stored in external (off-chip) read/write memory. The 6800 features seven addressing modes, including indexed addressing, and supports 72 instructions utilizing these modes. There are two hardware and one software initiated vectored interrupts, all of which automatically cause storage of the internal CPU's registers on the external stack for efficient interrupt handling. There is also a vectored restart function for power-up and system initialization. The 6800 requires only one +5 volt power supply and no external TTL devices for bus interface. The condition code register maintains five status bits and a master interrupt control bit.

The MC6800 is currently the second most widely used microprocessor. Originally designed as a second generation competitive device at the same time as the 8080, its design philosophy is evident in its unique architectural features. For instance, its strict adherence to the edict of no on-chip general purpose (scratchpad) registers is compensated for by the inclusion of a fast two-byte base page, direct addressing scheme. This allows the first 256 bytes of memory to be quickly accessed for data manipulation. The inclusion of two main accumulators facilitates fast "co-routine" handling and other interrupt driven task servicing. Its relative branch instructions are the same as those of the Digital PDP-11 minicomputer, and provide a very sophisticated decision making capability. On the negative side, however, the 6800 does pose some annoying

IA-52,853

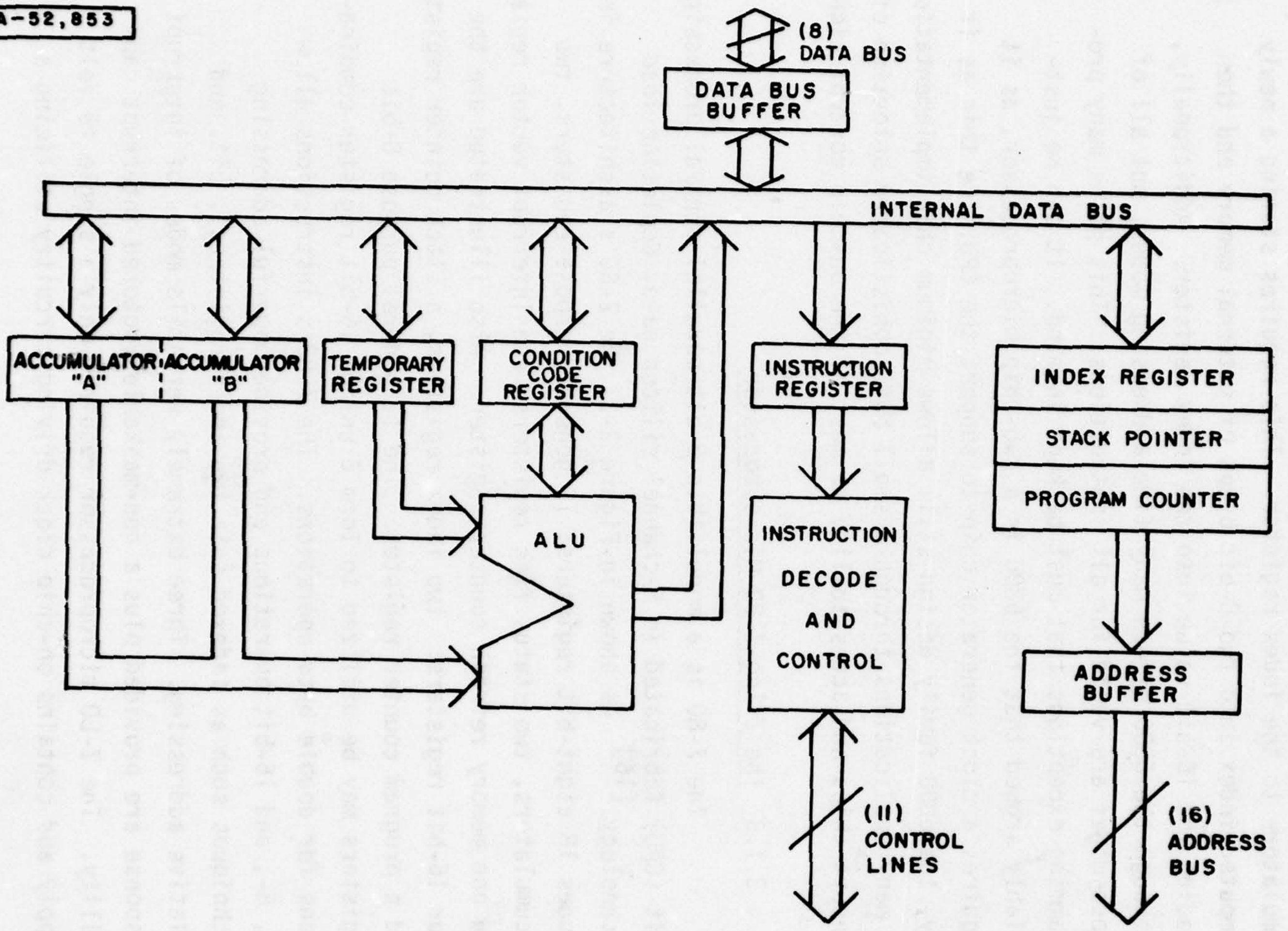
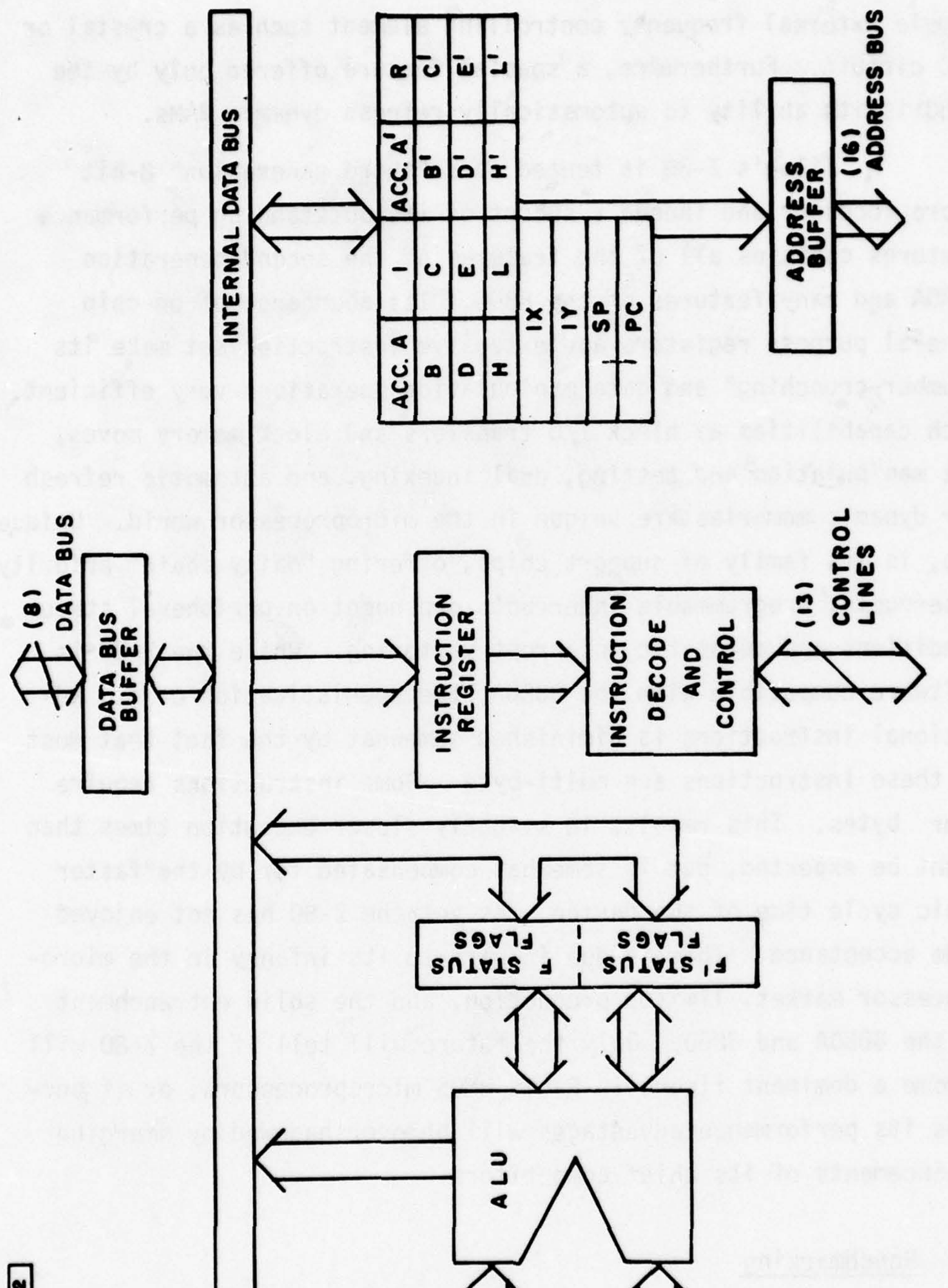


Figure 3-2 : 6800 MICROPROCESSOR ARCHITECTURE

problems. For example, there is no direct path from the two accumulators to the index register. This requires storing a newly computed index into two 8-bit bytes of external memory and then loading the 16-bit value into the index register. Additionally, although the 6800 offers many fine addressing modes, not all of these modes are valid for all instructions. This poses many programming exceptions that must be kept in mind. It can be justifiably argued that the 6800 is a two-chip microprocessor, as it requires a clock generator chip to support the CPU. Be this as it may, the 6800 family design still allows minimum chip implementation in many applications through a small but sophisticated selection of support chips and accessibility to most of the on-chip control signals.

3.1.3 The Zilog Z-80 Microprocessor

The Z-80 is a monolithic 8-bit parallel central processing unit (CPU) fabricated in N-channel silicon gate, depletion load technology.[16] As shown in Figure 3-3, the Z-80's architecture includes 18 eight-bit registers: 12 general purpose registers, two accumulators, two status flag registers, one interrupt vector register and one memory refresh counter register. Also illustrated are the four 16-bit registers: two index registers, a stack pointer register and a program counter register. The 12 general purpose 8-bit registers may be utilized to form 6 unique 16-bit register combinations for double byte operations. The Z-80's instructions allow 4-, 8-, and 16-bit operations and provide powerful addressing techniques such as indexed (via two index registers), bit, and relative addressing. Three extremely versatile modes of interrupt response are provided plus a non-maskable vectored interrupt capability. The Z-80 microprocessor requires only a single +5 volt supply and contains on-chip clock driving circuitry utilizing a



IA-52,852

Figure 3-3 Z-80 MICROPROCESSOR ARCHITECTURE

simple external frequency controlling element such as a crystal or R-C circuit. Furthermore, a special feature offered only by the Z-80 is its ability to automatically refresh dynamic RAMs.

Zilog's Z-80 is touted as a "third generation" 8-bit microprocessor and indeed a subset of its outstanding performance features contains all of the features of the second generation 8080A and many features of the 6800. Its abundance of on-chip general purpose registers and extensive instruction set make its "number-crunching" and data manipulation operations very efficient. Such capabilities as block I/O transfers and block memory moves, bit manipulation and testing, dual indexing, and automatic refresh for dynamic memories are unique in the microprocessor world. Unique too, is its family of support chips, offering "daisy-chain" priority interrupts, programmable interrupts dependent on peripheral status conditions and automatic interrupt vectoring. While the Z-80 is software compatible with the 8080A, the sophistication of its additional instructions is diminished somewhat by the fact that most of these instructions are multi-byte. Some instructions require four bytes. This results in slightly slower execution times than might be expected, but is somewhat compensated for by the faster basic cycle time of the device. As yet the Z-80 has not enjoyed wide acceptance. This is due in part to its infancy in the microprocessor market, limited production, and the solid entrenchment of the 8080A and 6800. Only the future will tell if the Z-80 will become a dominant figure in 8-bit NMOS microprocessors, or if perhaps its performance advantages will be overshadowed by emerging enhancements of its chief competitors.

3.2 Benchmarking

One of the most important factors in the microprocessor selection process is benchmarking. Benchmark programs can be common utility routines for general microprocessor comparison or specific program "kernels" or key routines critical to the task at hand. This paper adheres to the latter approach. Great care must be exercised when preparing benchmark programs. The same programmer should write the programs for all microprocessors under consideration, thus insuring the same level of coding expertise applied to each program. To further insure a fair comparison, the programmer should attempt to take advantage of the unique characteristics of each CPU. It is entirely possible that this will result in n-different programming approaches when writing n benchmark programs; no two approaches being exactly the same, and each fully exploiting strong points of the CPU to be utilized in the application. The results of such a benchmark program exercise should not be viewed as conclusive.[17] Benchmarking is not an end in itself; rather it is merely another tool to evaluate microprocessor performance under a specific, limited set of circumstances.

3.2.1 The Benchmark Program

A unique processing requirement for the (7,3) Reed-Solomon decoder application must be identified to serve as a "kernel" for benchmark programming. Recalling that decoder throughput is a very important design specification, the chosen kernel should represent the critical operation necessary to optimize this parameter. The MDD algorithm basically requires time consuming codeword table search and compare operations. These operations are normal utility routines for many types of data processors, including microprocessors. The unique characteristic of this procedure, however, is the fact that it is not enough to know that two codewords do not compare exactly, but we must have a measure of how badly they compare, i.e., the exact number of symbol discrepancies. A suitable choice for the benchmark program, therefore, is the "decoding kernel", i.e., the code table search and compare operation. Figure 3-4 is a flowchart of the functions included in this routine. Note that this program includes only the compare operation for one of the possible 512 valid (7,3) R-S codewords. The initialization assumptions inherent in this program are: 1) The table entry point using a systematic code (see A1.2.2) has already been calculated and loaded into the appropriate register(s), 2) The received codeword symbols have been loaded into the appropriate registers, and 3) The decision parameters for early table exit, table " wrap-around" addressing, and minimum distance determination have been previously calculated. The following paragraphs detail the three benchmark programs, specific initialization assumptions, register assignments, execution time, and program storage results.

3.2.2 Benchmark Program Analysis

In preparing the benchmark programs for the three candidate microprocessors, the prime objective was minimum execution time. While this objective does tend to minimize program storage, that result was definitely not vigorously pursued. Were minimum program storage the prime objective, a very tight program loop could have been written for each microprocessor. The resulting execution times would demonstrate a dramatic increase due to the overhead incurred by the conditional loop instructions. Indeed, one lesson learned early in the benchmarking effort was to employ "in-line" (no subroutine calls, or loops) code to achieve minimum execution time. Being fully cognizant of the programming strengths of each microprocessor, every effort was made to employ those strengths to speed execution time. If this philosophy necessitated a trade-off in memory for speed, then so be it, for it has long been a basic

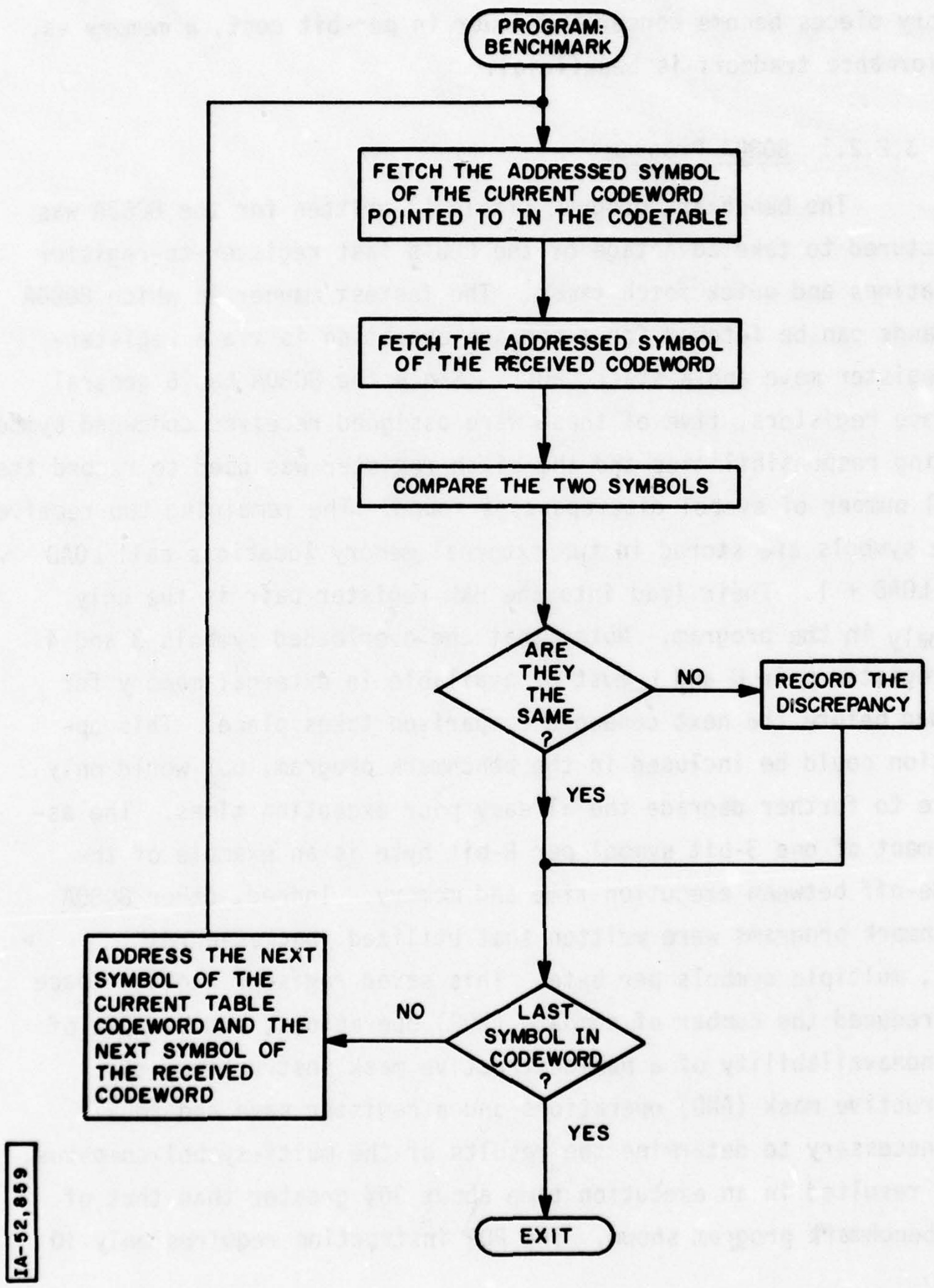

PROGRAM: BENCHMARK
FETCH THE ADDRESSED SYMBOL OF THE CURRENT CODEWORD POINTED TO IN THE CODETABLE
FETCH THE ADDRESSED SYMBOL OF THE RECEIVED CODEWORD
COMPARE THE TWO SYMBOLS
ARE THEY THE SAME ?
NO
RECORD THE DISCREPANCY
YES
LAST SYMBOL IN CODEWORD ?
NO
ADDRESS THE NEXT SYMBOL OF THE CURRENT TABLE CODEWORD AND THE NEXT SYMBOL OF THE RECEIVED CODEWORD
YES
EXIT
IA-52,859

Figure 3-4: BENCHMARK PROGRAM FLOWCHART

premise of the Low Cost Electronics project that as semiconductor memory pieces become denser and lower in per-bit cost, a memory vs. performance tradeoff is beneficial.

3.2.2.1 8080A Program

The benchmark program (Table I) written for the 8080A was structured to take advantage of the CPU's fast register-to-register operations and quick fetch times. The fastest manner in which 8080A operands can be fetched for a compare operation is via a register-to-register move and a stack "POP". Since the 8080A has 6 general purpose registers, five of these were assigned received codeword symbol holding responsibilities and the sixth register was used to record the total number of symbol discrepancies found. The remaining two received code symbols are stored in two external memory locations call LOAD and LOAD + 1. Their load into the H&L register pair is the only anomaly in the program. Note that the overloaded symbols 3 and 4 in register pair H and L must be available in external memory for reload before the next codeword comparison takes place. This operation could be included in the benchmark program, but would only serve to further degrade the already poor execution times. The assignment of one 3-bit symbol per 8-bit byte is an example of the trade-off between execution time and memory. Indeed, other 8080A benchmark programs were written that utilized "packed-words", i.e., multiple symbols per byte. This saved register storage space and reduced the number of compare (XOR) operations, but because of the nonavailability of a non-destructive mask instruction, two destructive mask (AND) operations and a register save and recall are necessary to determine the results of the multi-symbol compares. This resulted in an execution time about 30% greater than that of the benchmark program shown. The POP instruction requires only 10

TABLE I

8080A BENCHMARK PROGRAM

Initial Conditions:

Register B = Symbol Discrepancy Counting Register

Register C = Symbol Ø of the Received Codeword

Register D = Symbol 1 of the Received Codeword

Register E = Symbol 2 of the Received Codeword

Register H = Symbol 3 of the Received Codeword

Register L = Symbol 3 of the Received Codeword

CODE = Memory Location Containing Symbol 5 of the Received Codeword

CODE + 1 = Memory Location Containing Symbol 6 of the Received Codeword

- Received codeword symbols 3 and 4 are redundantly stored in external memory
- The stack poiner has been initialized to point to symbol #Ø of the valid stable codeword calculated from the table entry point routine
- Register B is clear (i.e., equal to zero)

PROGRAM:

Number of Machine States	Number of Program Storage Bytes	Instruction	Comments
10	1	POP PSW	fetch sym. #Ø of the table codeword
4	1	XRA C	compare with sym. #Ø of rec'd codeword
10	3	JZ 1	are the symbols the same?
5	1	INR B	no, record the discrepancy
10	1	1 POP PSW	
4	1	XRA D	SAME FOR SYMBOL #1
10	3	JZ 2	
5	1	INR B	
10	1	2 POP PSW	
4	1	XRA E	SAME FOR SYMBOL #2
10	3	JZ 3	
5	1	INR B	
10	1	3 POP PSW	
4	1	XRA E	SAME FOR SYMBOL #3
10	3	JZ 4	
5	1	INR B	

TABLE I (con't)

Number of Machine States	Number of Program Storage Bytes	Instruction	Comments
10	1	4 POP PSW	
4	1	XRA L	SAME FOR SYMBOL #4
10	3	JZ 5	
5	1	INR B	
16	3	5 LHLD CODE	Fetch symbols 5 & 6 of received codeword into on-chip registers L & H
10	1	POP PSW	
4	1	XRA L	
10	3	JZ 6	SAME FOR SYMBOL #5
5	1	INR B	
10	1	6 POP PSW	
4	1	XRA H	
10	3	JZ DONE	SAME FOR SYMBOL #6
5	1	INR B	

Execution Time T_{8080A} = 219(.5) = 109.5 microseconds

Number of program Bytes = 45

machine states to fetch an operand from external memory to the accumulator. This is a faster fetch method than the highly touted 8080A register-indirect fetch which requires 7 states plus 5 more to increment the register pair pointer for the next location. The disadvantage of utilizing the POP instruction which does not surface in the benchmark program, is the "housekeeping chore" of reinitializing the stack pointer for other segments of the complete decoder program. The second symbol of the compare operation is fetched by the XOR instruction in a fast register-to-register move (i.e., from the holding registers B through L to a temporary ALU operand register). This fetch and compare operation executes in the minimum 8080A instruction time, 4 machine states. A conditional jump instruction checks to see if the results of the XOR operation were zero (indicating identical symbols) or non-zero (indicating a symbol discrepancy). If there has been a symbol discrepancy, the B register is incremented (taking 5 machine state times) yielding a running total of how closely the two codewords compare. It is an idiosyncracy of the 8080A that it cannot perform a simple increment internal register operation in the minimum instruction execution time, unlike the 6800 and the Z-80. These operations continue repetitively until all the 8080A on-chip registers have been accessed. Because of the limited on-chip register storage, the last two symbols must be loaded over the two symbols in the H and L registers before the program can continue. As in all three benchmarks, there are no subroutine calls or repetitive loop instructions since the overhead time incurred in their execution would seriously increase program execution time. The results of the 8080A benchmark program show an execution time of 109.5 microseconds for a "kernel" program of 45 bytes in length. Remember, however, that before this kernel is executed for the next codeword in the table, the H & L registers must be reloaded with received symbols 3 and 4 and the B register cleared in addition to the common initialization assumptions of 3.2.1.

3.2.2.2 6800 Program

The benchmark program (Table II) written for the Motorola 6800 microprocessor takes advantage of two unique 6800 software instructions, and indexed addressing. The bit test instruction is an extremely powerful means of non-destructively testing the status of any bit pattern in an 8-bit word. It is ideally suited to determining the zero or non-zero results of a multi-symbol compare (XOR) operation without having to save, mask (AND), and refetch the accumulator contents. This allows "multi-symbol" packing of bytes and a decrease in the execution time of the compare operation. The "load direct" instruction is a fast two byte fetch instruction provided by the 6800 so that the lowest 256 bytes of memory may be treated as general purpose registers since the architecture of the 6800 does not provide any on-chip general purpose registers. This instruction facilitates the storage of the received codeword in RAM and still allows fast access to its symbols for the compare operations. Indexed addressing in conjunction with the XOR instruction is used to fetch a table codeword for the compare operation. Consider the program's three increment index register (INX) instructions. The observant programmer will recognize that this function could have been included more efficiently in the displacement byte of the three XOR indexed instructions reducing program storage and decreasing execution time. While this is certainly true, the reason for including these instructions is for fairness of comparison. This is because the other two benchmark programs use an updated stack pointer, which at the end of every codeword compare operation, immediately points to the next codeword in the table. In the 6800 program, were the INX instructions

TABLE II

6800 BENCHMARK PROGRAM

INITIAL CONDITION:

Memory Location CØ = symbol Ø & 1 of the received codeword
Memory Location C1 = symbol 2 & 3 of the received codeword
Memory Location C2 = symbol 4 & 5 of the received codeword
Memory Location C3 = symbol 6 of the received codeword

(All memory addressed ≤ 255 in RAM)

- Accumulator B = Symbol Discrepancy Counting Register
- Accumulator B is clear (i.e., equal to $ØØ_H$)
- The index register contains the address of the first byte of a table codeword which was chosen as the first comparison word by a table entry point subroutine.

PROGRAM:

Number of Machine Cycles	Number of Program Storage Bytes	INSTRUCTION	COMMENTS
3	2	LDAA CØ	fetch symbols Ø & 1 of rec'd codeword
5	2	EORA X	compare with symbols Ø & 1 of table codeword;
2	2	BITA #$ØF_H$	mask out symbol 1 result;
4	2	BEQ NXT	are Ø symbols equal?
2	1	INCB	no, record discrepancy
2	2	NXT BITA #$FØ_H$	mask out symbol Ø result
4	2	BEQ NXT1	are 1 symbols equal?
2	1	INCB	no, record discrepancy
3	2	NXT1 LDAA C1	fetch 2 & 3 of rec'd codeword
4	1	INX	point to next byte in codetable
5	2	EORA X	.
2	2	BITA #$ØF_H$	.

TABLE II (CON'T)

Number of Machine Cycles	Number of Program Storage Bytes	INSTRUCTION	COMMENTS
4	2	BEQ NXT2	.
2	1	INCB	(Same for remaining symbols)
2	2	NXT2 BITA #FØ$_4$	.
4	2	BEQ NXT3	.
2	1	INCB	. .
3	2	NXT3	.
4	1	INX	.
5	2	EROA X	.
2	2	BITA #ØF$_H$	.
4	2	BEZ NXTR	.
2	1	INCB	.
2	2	NXT4 BITA #FØ$_H$	.
4	2	BEQ NXT5	.
2	1	INCB	.
3	2	NXT5 LDAA C3	.
4	1	INX	.
5	2	EORA X	.
4	2	BEQ DONE	
2	1	INCB	
98	52	DONE	

Execution time T_{6800} = 98 microseconds

Number of program bytes = 52

not included in the benchmark,they would still have to be executed in preparation for the next codeword compare. Thus, their exclusion would present an unfair advantage to the 6800 program. Another instruction set advantage which the 6800 offers over the other two microprocessors is its set of conditional branch instructions. Although not fully utilized in the benchmark program, these instructions prove invaluable in the decision making process that follows every codeword compare operation. The 6800's second accumulator can be utilized as an on-chip register. In this program it serves as the symbol discrepancy counting register allowing increments in 2 machine cycles, the minimum instruction execution time. The results of the 6800 benchmark program show an execution time of 98 microseconds for a "kernel" program of 52 bytes.

3.2.2.3 <u>Z-80 Program</u>

The Z-80 benchmark program (Table III) is analagous to the 8080A program. Indeed, since the Z-80 is software compatible with the 8080A, it could execute the same program. The only advantage in doing this would be the decrease in execution time resulting from the Z-80's basic 400 ns machine state time vs. the 8080A's 500 ns. To further decrease the execution time of the program we can take advantage of two Z-80 characteristics. Since the Z-80 has twice as many scratch-pad registers as the 8080A, all of the received codeword symbols can be stored on-chip and manipulated with fast register-to-register moves. Also, the increment register instruction in the Z-80 is an instruction which executes in the minimum instruction execution time (4 states) as opposed to 5 in the 8080A. The disadvantage of storing all symbols on-chip is that the Z-80 technique requires the inclusion of an "EXCHANGE" register instruction. Furthermore, once the back switching has occurred, the

TABLE III

Z-80 BENCHMARK PROGRAM

INITIAL CONDITIONS:

Register B = symbol Ø of the received codeword
Register C = symbol 1 of the received codeword
Register D = symbol 2 of the received codeword
Register E = symbol 3 of the received codeword
Register H = symbol 4 of the received codeword
Register B' = symbol 5 of the received codeword
Register C' = symbol 6 of the received codeword
Register L = symbol Discrepancy Counting Register
Register L' = symbol Discrepancy Counting Register

Register L & D' are clear (i.e., equal to $ØØ_H$)

The stack pointer has been initialized to point to symbol #Ø of a table codeword which was chosen as the first comparison word by a table entry point subroutine.

PROGRAM:

Number of Machine States	Number of Program Storage Bytes	INSTRUCTION INSTRUCTION	COMMENTS
10	1	POP AF	fetch symbol #0 of table codeword
4	1	XOR B	compare with symbol #0 of received codeword
10	3	JZ NXT	are symbols the same?
.4	1	INC L	No, record discrepancy
10	3	NXT POP AF	
4	1	XOR C	
10	3	JZ NXT1	SAME FOR SYMBOL #1
4	1	INC L	
10	1	NXT 1 POP AF	
4	1	XOR D	
10	3	JZ NXT2	SAME FOR SYMBOL #2
4	1	INCL	

TABLE III (CON'T)

Number of Machine States	Number of Program Storage Bytes	INSTRUCTION	COMMENTS
10	1	NXT 2 POP AF	
4	1	XOR E	
10	3	JZ NXT3	SAME FOR SYMBOL #3
4	1	INC L	
10	1	POP AF	
4	1	XOR H	SAME FOR SYMBOL #4
10	3	JX NXT4	
4	1	INC L	
4	1	NXT 4 EXX	Switch register banks
10	1	POP AF	
4	1	XOR B'	SAME FOR SYMBOL #5
10	3	JX NST5	Note: discrepancy register is now D'
4	1	INC D'	
10	3	NXT 5 POP AF	
4	1	XOR C'	SAME FOR SYMBOL #6
10	3	JZ NXT6	
4	1	INC D'	
4	1	NXT 6 LDA D'	combine both discrepancy counts
4	1	EXX	
4	1	ADD L	
212	46		

Total Program execution time T_{Z-80} = 212(.4) = 84.8 microseconds

Total number of program storage bytes = 46

L register, which was used to accumulate the total number of symbol discrepancies found, is not directly addressable. So a new register in the other bank must be utilized to record the number of symbol discrepancies for the last two symbols. Again, for fairness of comparison, instructions must be included in the benchmark program to add these two registers to arrive at a total symbol discrepancy number. It should be noted that although Z-80 architecture offers two separate index registers, the fetch instructions utilizing indexed addressing are much slower executing than the chosen stack pops and register-register moves. Also, it is highly regrettable for this application at least, that the Z-80 does not provide any instruction equivalent to the 6800's bit test. Were such a non-destructive mask and check instruction available, a 16-bit subtract (i.e., a 4 symbol compare) could be utilized for multi-symbol comparison (as suggested in 2.2.1) and a dramatic savings in execution time, and program storage would result. The bit instructions of the Z-80 although highly capable for individual bit sensing and control, do not provide a multi-bit,masking capability like the 6800's Bit Test, and are not directly applicable to multi-bit symbol operations. The results of the Z-80 benchmark program show an execution time of 84.8 microseconds for a "kernel" program of 46 bytes.

3.3 Key Factors Comparison

Eighteen key factors from the design requirements analysis of Section 2.0 have been selected for their applicability to the (7,3) R-S decoder application. Table IV is a microprocessor comparison table which shows the results of a comparison study between the three candidate microprocessors based on these 18 key factors. The number assigned under each microprocessor for every factor is an indicator of the microprocessors' ability to satisfy a particular

TABLE IV

MICROPROCESSOR COMPARISON

FACTORS	INTEL 8080A	MOTOROLA 6800	ZILOG Z-80
SYSTEM			
1. Software Support	3	3	3
2. Documentation	3	3	2
3. Application Notes	2	3	1
4. Design Aids	2	2	2
5. Second Source	3	2	1
6. Availability	2	2	2
7. MIL-Qualification	2	2	1
8. Benchmark	1	2	3
SOFTWARE			
9. Word Size	0	0	1
10. Addressing	1	2	3
11. Internal Registers	2	0	3
12. Instruction Set	1	2	3
HARDWARE			
13. Clock	1	1	2
14. Chip Voltages	0	2	2
15. Interrupts	1	2	3
16. I/O & Memory Expansion	2	2	3
17. Minimum Chip Count	0	1	2
18. Support Chips	3	2	1
TOTALS	29	33	38

design requirement. The numbers range from 0 to 3 and have been assigned according to the following rules:

0 = the microprocessor in no way satisfied the design factor as discussed in Section 2.0

1 = the microprocessor only partially satisfied the design factor as discussed in Section 2.0

2 = the microprocessor completely satisfied the design factor as discussed in Section 2.0

3 = the microprocessor's capability exceeds that required by the design factor as discussed in Section 2.0.

The rationale for the specific assignments of these numbers is detailed in the following paragraphs.

3.3.1 System Factors Rationale

Software Support - All three microprocessors are supported by extensive software products. Each manufacturer offers and supports a broad line of system programs including language translators for higher-level languages, debug packages with trace routines, and in-circuit emulation capability.

Documentation - Motorola and Intel are considered to have excellent microprocessor documentation. Both products are supported by plentiful, well-organized design manuals. Zilog documentation is also quite satisfactory, but being a more recent product line has not yet enjoyed the benefit of a myriad of review and revision cycles.

Application Notes - Motorola with its mammoth application manual has more than initiated and educated its user community. The manual is an excellent system introduction, a detailed design document, and includes many specific application examples with typcial sample programs. Intel application notes were judged very satisfactory especially when bolstered by numerous noncompany-related manuals and application articles on the evolutionary 8008-8080A. Zilog Corporation's application notes are few and specific. Instead, Zilog is counting on converting 8080A and 6800 users and concentrates on demonstrating how the Z-80 can better perform applications well-known to the other two microprocessors.

Design Aids - All three microprocessor manufacturers offer a rather complete line of prototype boards, chip sets, and sophisticated development systems. Even where deficiencies in a design aid line may exist, they are quickly corrected when considering the wealth of design aid products marketed by non-affiliated companies.

Second Source - Intel is judged to be in the strongest second source position having its widely accepted 8080A supported by Advanced Micro Devices, Texas Instruments, Nippon Electric Company, National Semiconductor, Hitachi and Siemens. Motorola enjoys sizeable second sourcing by American Microsystems International, Fairchild, Hitachi and Sescosem/Thomson-CSF. Zilog is in the weakest second source position with Mostek currently supplying the majority of available pieces.

Availability - Pieces are readily available factory direct or through distribution for all the microprocessors.

MIL-Qualification - As discussed in 3.1.4, Intel and Motorola parts are currently being Air Force specified under

MIL-M-38510 while Zilog only offers parts which it will guarantee to MIL-STD-883B but have not been DOD specified.

Benchmark Program - As detailed in Section 3.2.2, the benchmark results must be analyzed very carefully. Although all these microprocessors can perform the decoding function, the Z-80 execution time is faster than the 6800 or 8080A, while the number of program storage (bytes) required is about the same.

3.3.2 Software Factors Rationale

Word Size - As discussed in 2.2.1, the optimum CPU word size for the R-S decoder should be 16 bits. However, 16-bit NMOS microprocessors are relatively new to the market and few have been MIL-qualified as yet. The 8-bit NMOS microprocessor, however, is the most prolific member of the microprocessor family. Consequently, it is enjoying large volume production and attendant price decreases. Design enhancements, multiple sourcing, and military qualification are all recent characteristics of this ubiquitous piece, and further justify its use in the decoder application.

Since none of the microprocessors under consideration are 16-bit CPU's, they do not satisfy the word size requirement. The Z-80, however, does possess some 16-bit load and arithmetic instructions and therefore minimally approaches 16-bit operational capability.

Addressing - As illustrated in Table V, the Z-80 offers the most sophisticated addressing modes, particularly bit addressing, 13 registers for indirect addressing, and two registers for indexed addressing. The 6800 also offers indexed addressing, but through only 1 register and no indirect capability. Its two byte "direct" form, however, is a particularly quick fetching mechanism. The 8080A offers the least powerful addressing capability of this group.

TABLE V

MICROPROCESSOR ADDRESSING MODES

	8080A	Z-80	6800
1.	Direct (3 bytes)	Extended (3 bytes)	Extended (3 bytes)
2.	------	-----	Direct (2 bytes)
3.	Register	Register	-----
4.	Register Indirect (6 registers)	Register Indirect (12 registers)	-----
5.	-----	Relative	Relative
6.	-----	Indexed (2 registers)	Indexed (1 register)
7.	Implied	Implied	Inherent
8.	Immediate	Immediate	Immediate
9.	-----	Bit	-----

Internal Registers - The Z-80 exceeds this design desirable by providing 14 internal registers, while the 8080A has 6 and the 6800 none. (Note: These are in addition to accumulator. processor status word, stack pointer, program counter or index registers).

Instruction Set - As noted in the benchmark program analysis (see Section 3.2), the absence of one instruction in the Z-80 and the inclusion of that instruction in the 6800 made a tremendous difference in the suitability (and software approach) of the CPU to the "decoding kernel" benchmark routine. Nevertheless, considering the entire application program requirements, the Z-80 instruction set was judged more powerful than the 6800's. The 8080A's instruction set, being a subset of the Z-80's, is less powerful, and although the 8080A includes on-chip registers, its instruction set was judged not as powerful as that of the 6800 for this particular application.

3.3.3 Hardware Factors Rationale

Clock - The 8080A and 6800 microprocessors both require multi-phase external clock generator/drivers. The Z-80 possesses on-chip generator/driver circuitry and only requires a crystal, frequency determining R-C circuit, or TTL level, single-phase, clock signal. The clock period of the standard Z-80 is faster than the other two standard parts.

Chip Voltages - The 8080A requires three different voltages (± 5 and 12 volts), while the Z-80 and 6800 CPU's require only a +5 volt supply.

Interrupts - The 8080A provides limited organic interrupt handling capability, requiring complex external support for interrupt arbitration, identification, and vectoring. The 6800 provides two hardware (maskable and non-maskable), and one software interrupts (all vectored) which can be modified with external logic for additional capability. The Z-80 provides 3 distinct interrupt modes (under software control) including the 8080A scheme, the 6800 maskable and non-maskable capability, and its own special vectored scheme. Additionally, the Z-80 structure allows "daisy-chaining" the interrupts of peripheral devices for simple arbitration, identification and prioritizing.

<u>I/O and Memory Expansion</u> - All three microprocessors offer adequate I/O memory expansion. The 6800, exhibiting "memory-mapped" I/O, has inherently more I/O expansion capability than the programmed I/O of the Z-80 or 8080A. However, these two CPU's can be modified to provide memory mapped I/O by the inclusion of a few simple external gates. The Z-80 greatly facilitates the use of large external memory by providing automatic refresh capability for dynamic memory chips.

<u>Minimum Chip Count</u> - The 8080A is in reality a three chip microprocessor requiring an external system controller chip and clock generator/driver chip in addition to the CPU. The 6800 always requires an external multi-phase clock generator/driver chip or equivalent discrete circuitry. The Z-80 only requires one or two external passive components for CPU operation. All three microprocessors may or may not require buffers and I/O chips in addition to program storage ROM for minimum chip systems.

<u>Support Chips</u> - Intel's 8080A probably has the most capable selection of support chips available to increase its performance capability, followed closely by Motorola's M6800 family. Zilog's Z-80 being a relatively new entry in the field has an adequate but not extensive support chip family. This comparison factor can be quickly voided when allowances are made for utilizing support chips

designed for one processor family in a design incorporating a different microprocessor. The industry is currently beginning to supply general purpose support chips (A/D/A, I/O controller, etc.) which may be interfaced with all of the popular microprocessors.

4.0 CONCLUSIONS

The results of the benchmark programs indicate that the Zilog Z-80 microprocessor is the best hardware implementation choice for the (7,3) R-S decoder. However, on closer examination, the results are not clearly indicative. Table VI is a summary of the benchmark program results. Since the main objective of the design is to maximize throughput, execution time must be kept to a minimum. The Z-80's 84.8 microseconds was the fastest running routine, being 13.5% faster than the 6800 program. However, when it is considered that the Z-80's clock cycle of 400 ns is 20% faster than the 500 ns clock cycle of the 6800, the 6800 program is actually more efficient. Further proof of this fact is evidenced by the number of storage bytes required for the codetable. The Z-80's 3574 bytes is over twice the codetable storage required by the 6800 program, while the program storages are within 6 bytes, i.e., 13% of each other. Indeed, both of these results illustrate the advantages gained from the efficiency of the non-destructive mask operations provided by the 6800 instruction set as discussed in 2.2.4.

As shown in Table IV, the Z-80's comparative factor total of 38 is 5 points better or 15% higher than that of the 6800 and 31% higher than the cumulative rating of the 8080A. It is interesting to look at the results of Table IV in a slightly different manner. Grouping the 18 comparative factors into the three major categories of System, Software, and Hardware, and subtotalling each, results in a comparison structured as shown in Table VII. As this table illustrates, the Z-80 scores quite poorly with respect to system factors, is clearly superior in software ratings and appears only marginally better in the hardware comparison. Although such features as on-chip registers and sophisticated addressing and instructions are provided by the

TABLE VI

(7, 3) R-S Decoder Benchmark Program Results

	8080A	6800	Z-80
EXECUTION TIME (microseconds)	109.5	98	84.8
PROGRAM SIZE (bytes)	45	52	46
CODETABLE STORAGE (bytes)	3574	1536	3574

TABLE VII

MICROPROCESSOR COMPARISON SUMMARY

	INTEL 8080A	MOTOROLA 6800	ZILOG Z-80
SYSTEM FACTORS	18	19	15
SOFTWARE FACTORS	4	4	10
HARDWARE FACTORS	7	10	13
TOTALS	29	33	38

Z-80, they are not all used in the benchmark routine and consequently do not produce the vastly superior performance that Table IV might lead one to believe.

It must be concluded, based on the benchmark program results and the table of comparison factors, that the Z-80 microprocessor is the best candidate for the decoder when speed of execution is emphasized. However, it is also true that the speed advantage is not as great as would be expected from the Z-80's faster clock speed alone, and therefore must be attributed to less efficient coding (i.e., instruction set suitability). This is further evidenced by the much larger codetable storage requirements of the Z-80 program over that of the 6800.

It must also be noted that the comparative results of Tables IV through VII are extremely time-dependent. With the rapid evolution of new and improved microprocessors, other candidates might be considered. Indeed, CPU's which are design enhancements of the considered devices and which correct many of the deficiencies pointed out in Section 3.0 such as the 6801, 6802, 6809, 8085, 8748, Z-8000, MC-68000, 8086, etc., are very promising alternatives. However, at the time of this work and because of considerable in-house design experience and the availability of support software and demonstration/ debug hardware for the Motorola MC6800 microprocessor, it was decided to implement the (7,3) R-S decoder using that device. The small speed advantage offered by a standard Z-80 microprocessor-based decoder was deemed not significant enough to warrant the additional hardware effort necessary to implement that device. The detailed design test and evaluation of the 6800-based (7,3) Reed-Solomon decoder is described in Volume II of this report.

APPENDIX

FUNDAMENTALS OF ERROR CORRECTION CODING

A1.0 History

In 1948, Claude Shannon published his now famous discrete noisy channel coding theorem.[4] This theorem states that for any rate R less than the channel capacity C, and for any $\varepsilon > 0$ there exists for all suitably large N, a code of rate $R = k/N$ for which the probability of a decoding error [P(e)] is less than ε when a maximum-likelihood decoder is used. This event started the academic community on a search for the promised codes. Unfortunately, Shannon's work did not explicitly reveal how such codes could be obtained, it only proved their existence. Consequently, the field of error correction coding as it exists today has taken many diverse paths. Basically, however, code development has evolved two types of codes. Block codes treat message sequences independently, associating a N-tuple block of channel symbols with each possible k-tuple block of information symbols. The resultant codewords are transmitted, corrupted by channel noise, and decoded independently of all other received codewords. Tree codes, on the other hand, process information continuously, associating message sequences with code sequences on the basis of the present message and of previously encoded messages. The best known Tree codes are a subclass called Convolutional Codes.

Block codes have been quite extensively analyzed in the 25 years since Shannon's famous article. These codes can be rigorously associated with many algebraic and geometric concepts. The algebraic foundation of many "strong" (i.e., powerful multiple error-correcting) codes facilitates reasonable implementation of encoding and decoding circuitry. Initial investigations undertaken as part of MITRE's Low Cost Electronics project (7010) have concentrated largely on block code application studies and the realization of low-cost decoding hardware for these codes.

A1.1 Finite Field Arithmetic

An algebraic system is termed a finite field if it contains a finite set of elements for which the operations of multiplication, addition, and their inverses are defined, and if every non-zero element in the set has a multiplicative inverse. Considering a finite binary field of two elements (0, 1), addition and multiplication can be defined modulo-2. Generally, a modulo-q reduction is expressed in a congruence relationship where an integer Z in the finite field is said to be congruent to E modulo-q written $Z \equiv E \bmod q$ if and only if, by the Euclidean Algorithm,[1] $Z = Dq + E$, where $0 \leq E < q$. In the binary field $q = 2$. In general a field can be defined over any number q which is a positive power of a prime number, i.e., $q = P^m$. This is termed a Galois Field, GF(q) or GF(p^m). If $m = 1$, the field is called a base or ground field. All algebraic operations in the base field are defined modulo-p. If $m > 1$, the field is termed an m-extension field of characteristic p and all algebraic operations are defined modulo-p(x), an m^{th} degree polynomial having coefficients in the base field. The q field elements of GF(q) can be expressed as m-tuples over the base field GF(p). The m^{th} degree modulo-polynomial p(x) that defines the arithmetic operations in GF(p^m) is indivisible by any other polynomial of degree less than m and greater than 0, for this reason it is termed an irreducible polynomial. Furthermore, all non-zero elements of GF(q) can be generated by utilizing one element of the field (α) called a primitive element, where $p(\alpha) = 0$, and successively calculating powers of this m-tuple element reduced modulo-$p(x)$. It will be found that all of the $q-1$ non-zero elements of $GF(q)$ can be generated in this manner before any element repeats. It would seem then that if an m^{th} degree polynomial $p(x)$ with coefficients over $GF(p)$ is wisely selected, it should be possible to uniquely generate all the elements of $GF(p^m)$. This is indeed the case and such a polynomial is called a <u>primitive irreducible polynomial</u>.

A1.2 <u>Linear Block Codes</u>

Consider a k-digit information message sequence, where each digit is any field element of $GF(s)$. There are, therefore, s^k unique message sequences we may wish to transmit. If we isomorphically assign each information message sequence, to an N-digit, $N \geq k$, channel "transmission" message sequence, where each channel digit is any field element of $GF(q)$, where $q^N > s^k$, we have a set of size $M = s^k$ q-ary N-tuple channel "codewords". Any such M sequence set of q-ary N-tuples so assigned is called a <u>block code</u>. Alternatively, if a k-length sequence of information digits is mapped into an N-length sequence of channel digits, such a block of channel digits is called a codeword, and the finite set of codewords resulting from the mapping of a finite set of k-length information sequences comprises a block code. If our isomorphic mapping is a linear transformation we say the resultant code set is a <u>linear block code</u>. The nature of the transform employed profoundly affects the structure of the code and exploiting the algebraic structure of this transform leads to efficient encoding and decoding methods.

A1.2.1 <u>Cyclic Codes</u>

Consider a block code of q-ary N-tuples as comprising a vector space M_N, wherein each code vector $\underline{S} = (s_0, s_1, \ldots, s_{N-1})$

contains N elements and each element can be any member of GF(q). If any vector $S = (s_{N-1}, s_0, s_1, \ldots, s_{N-2})$, formed by cyclically shifting the components of $\underline{S}$ is also in M_N, then such a set of cyclically shifted code vectors is called a linear cyclic code. The N-length codewords over GF(q) of a linear cyclic code (vectors of the vector space M_N) can be associated with the algebra of residue classes of polynomials modulo X^N-1.[5] Under such an association, we can represent the information sequences, codewords, and even the isomorphic transformation between them in polynomial form, i.e., corresponding to every N-tuple $(a_{N-1}, a_{N-2}, \ldots, a_0)$ there is a polynomial of degree N-1 or less, e.g., $C(x) = a_{N-1} X^{N-1} + a_{N-2} X^{N-2} + \ldots + a_0$. Consider the least-degree codeword N-tuple polynomial in a linear cyclic code as $C'(x) = c_0 + c_1 X + c_2 X^2 + \ldots c_{r-1} X^{r-1} + c_r X^r$, i.e., an r^{th} degree polynomial, were $c_i = 0$, $r < i, < N-1$. Call this least degree polynomial g(x) instead of $C'(x)$. Since the code is a cyclic code, the code polynomials $Xg(x), X^2g(x), \ldots, X^{N-r-1} g(X)$ (i.e., shifts of g(x)), are also codewords. Since the code is a linear cyclic code, any linear combination of these codewords must also be a codeword, or

$$\begin{aligned} C(x) &= W_0 g(X) + W_1 Xg(X) + \ldots + W_{N-r-1} X^{N-r-1} g(X) \\ &= (W_0 + W_1 X + \ldots + W_{N-r-1}) g(X) \\ &= W(X) g(X) \qquad (1) \end{aligned}$$

Therefore, the message polynomial may be defined as W(x), the codeword as C(x), and the transform or generator polynomial as g(x). Since the objective is to create an (N,k) linear cyclic code, associating an N-tuple code vector with C(x) and a k-tuple message vector with W(x), requires that the degree of the code polynomial C(x) be N-1, the degree of the desired message polynomial be k-1, and the degree of the generator polynomial must therefore be N-k.

Equation (1) illustrates that any cyclic code polynomial can be evenly divided (i.e., without a remainder) by its generator polynomial g(x). Define $\alpha_1, \alpha_2, \alpha_3 \ldots, {}_{N-1}$ (as possibly distinct) roots of c(x) over GF(q), then $c(\alpha_1) = 0$ for $1 \leq 1 \leq N-1$. Define the <u>minimum polynomial</u> of α_i as the monic polynomial $m_i(x)$ of smallest degree with coefficients over GF(p), (the base field), such that $m_i(\alpha_i) = 0$. By Euclid's Algorithm, $c(x) = m_i(x)q(x) + r(x)$ and since $c(\alpha_i) = 0$ and $m_i\ (\alpha_i) \underline{\Delta}\ 0$, $r(\alpha_i)$ must be equal to zero, and therefore $m_i(x)$ divides c(x). Since all the minimum polynomials $m_i(x)$ divide c(x), their least common multiple must also divide c(x) and therefore:

$$g(x) = \text{LCM} \left\{ m_1(x), m_2(x), \ldots\ldots\ m_N(x) \right\} \qquad (2)$$

A1.2.2 <u>BCH Codes</u>

The set of codes devised by Bose, Chaudhuri, and Hocquenhem (BCH codes) is a generalized case of the well known binary Hamming codes. Further generalized to codes in p^m symbols by Gorenstein and Zierler, these codes were designed to correct additive errors on symmetric, one-way, memoryless, Hamming-metric channels which employ orthogonal signalling.[6] It has been shown that insofar as such channels approximate "real-world" communications paths, employing BCH codes results in significant improvements in channel reliability.

BCH codes are formally defined as follows: if α is a field element of $GF(q^m)$, choose any $m_0 > 0$ and any d ($2 \leq d < N$), and form the d-1 powers of $\alpha^{m_0}, \alpha^{m_0+1}, \ldots\ldots, \alpha^{m_0+d-2}$. The lowest degree generator polynomial $g(x)$ over $GF(q)$ which contains these powers of α as its roots generates a q-ary BCH code, with block length $N = q^m - 1$. The generator polynomial of this q-ary BCH code, being a linear cyclic code, is similar to equation (2), or:

$$g(x)_{BCH} = LCM \left\{ m_{m_0}(x), m_{m_0+1}(x) \ldots\ldots, m_{m_0+d-2}(x) \right\} \quad (3)$$

where $m_{m_0+i}(x)$ $0 \leq i \leq d-2$, are the minimum polynomials of the d-1 roots. We know that in order to correct t or fewer symbol errors in a (N,k) q-ary BCH code, the minimum Hamming distance d between code-words must be related to t by $2t = d-1$. Using this relationship and setting $m_0 = 1$ in equation (3)

$$g(x)_{BCH} = LCM \left\{ m_1(x)\ m_2(x), \ldots\ldots, m_{2t}(x) \right\} \quad (4)$$

The q^m-1 non-zero elements of $GF(q^m)$ comprise a finite set. Consequently, if we form the powers of one element of the field α, as α^i, then these values must begin to repeat at some power $i = j$. The smallest positive value of j where this occurs is called the <u>order</u> of the element. The degree of any of the minimum polynomials $m_i(x)$ in (4) is the least integer ℓ such that the order of α^i divides $q^\ell - 1$. All $m_i(x)$ are irreducible, since these polynomials have no factors with coefficients in $GF(q)$. The order of the roots of an irreducible polynomial is called the <u>exponent</u> e to which that polynomial belongs. If an irreducible polynomial belongs to e then it divides X^e-1 but no polynomial of the form X^n-1 for any $n < e$. If α

is chosen to be a primitive element of $GF(q^m)$, i.e., it is possible to generate all of the q^m-1 non-zero elements of $GF(q^m)$ from powers of α, then its order is q^m-1. Alternatively, the maximum order of an element of a field is achieved when that element is primitive. The minimum polynomial $m_1(x)$ in (4) derived from the primitive element α, is called an Irreducible, Primitive Polynomial. Since one of its roots is primitive, all of its roots are primitive and of the same order, i.e., $m(x)$ belongs to q^m-1. The degree of $m_i(x)$ is therefore m and from (4) it can be seen that the degree of $g(x)$ and the number of parity check symbols is $N-k \leq 2mt$.

In Section A1.2.1 it was shown that all of the minimum polynomials $m_i(x)$ of (2) divide the code polynomials. Therefore, the degree of the code polynomials (N) must be the Least Common Multiple (LCM) of the exponents to which the minimum polynomials $m_i(x)$ belong. This is equivalent to the LCM of the orders of the roots of $m_i(x)$. As we have previously shown, the maximum order of an element results when that element is a primitive element or q^m-1, therefore $N = q^m-1$. More generally, since all of the roots of $g(x)$ are also roots of $C(x)$, the length of the code is the least common multiple of the orders of the roots. This is because the elements of GF (q^m) form a multiplicative group wherein the order of any element divides the order of the group, and the least common multiple of such a group must be q^m-1. Codes formed in this manner, with α taken as a primitive element of $GF(q^m)$ are called "primitive q-ary BCH" codes.

By far the most common of all q-ary BCH codes are the binary codes where $m_0 = 1$, $p = 2$, and $m = 1$. As a consequence of these parameters, for any positive integer m and t (where $t < N/2$) there exists a binary BCH code of:

Code Length: $N = 2^m - 1$

No. of Parity Check Digits: $N - k \leq mt$

and $g(x) = \text{LCM} \left\{ m_1(x), m_3(x), \ldots, m_{2t-1}(x) \right\}$

Binary BCH codes have been widely analyzed and published. Our efforts, however, will concentrate on non-binary BCH codes, and in particular a special subclass, the Reed-Solomon codes.

A1.2.3 Reed-Solomon Codes

In Section A1.2.2 it was shown that for a primitive q-ary BCH code, a primitive element α was defined over an m-dimensional extension field $GF(q^m)$, and coefficients of the generator polynomial $g(x)$ were defined over the base field $GF(q)$. If $m = 1$, the extension field over which the primitive element is defined is the same as the base field for the code elements. A q-ary BCH code of this type is called a Reed-Solomon code. Since α is primitive its order is $q - 1$ and therefore the code length is $N = q - 1$.

Because the extension field and the base field are equivalent, the minimum polynomials $m_i(x)$ of (4) can be simply expressed as: $m_i(x) = X - \alpha^i$. The generator polynomial of an (N,k) q-ary R-S code then becomes

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3) \ldots (x - \alpha^{2t}) \qquad (5)$$

It is obvious from (5) that the degree of $g(x)$ is 2t and therefore the number of parity check digits is:

$$N - k = 2t = d - 1 \qquad (6)$$

Manipulating equation (6), it can be seen that the Hamming distance d of an (N,k) R-S code is N - k + 1. Codes exhibiting this distance are called "<u>Maximum Distance Separable</u>". That this minimum distance is indeed "maximum" can be shown by recalling that the minimum distance of a code is equal to the minimum weight of any non-zero codeword. In any systematic R-S code there exist codewords with only one non-zero information symbol. In addition to this one non-zero symbol, there are only N-k remaining symbols that can be non-zero, so the "maximum-minimum" weight of such words can be no greater than N - k + 1. Note that the maximum distance separable equation,

$$d = N - k + 1 \qquad (7)$$

is satisfied for only two trivial cases of binary linear codes: (a) repetition codes, where k = 1 and d = N and, (b) single parity-check codes, where k = N - 1 and d = 2. Therefore, R-S codes, unlike any non-trivial binary codes provide maximum distance properties.

The block length of any R-S code is N = q - 1 and therefore an (N,k) code can be thought of as a (q-1, q-1-2t) code.

If the field size q is a power m of a prime number p, i.e., $q = p^m$, then we can define binary, non-binary, base field and extension field versions of R-S codes. The q-ary symbols of such codes can be considered as m-tuples over GF(p) and a t-symbol error correcting (P^m-1, P^m-1-2t) R-S code over $GF(p^m)$ can be regarded as a $(m(P^m-1), m(P^m-1-2t))$ code over GF(p). Viewed in this way it is obvious that the code is capable of correcting any error pattern whose non-zero digits are confined to t m-symbol blocks.

Initial hardware efforts in microprocessor implementations of short error correcting codes have been focused on a simple binary extension field multiple-error correcting (7,3) R-S code. The coding advantages of R-S codes have been previously shown. In particular, the binary extension field (7,3) R-S code was chosen for 3 reasons:

1. There is current interest in (7,3) coding for anti-jam protection in our DME data link work (Project 90020).
2. A short code like the (7,3) allows simple maximum-likelihood decoding schemes to be employed.
3. A code over GF(2^3) allows the realization of efficient arithmetic operations in current MOS microprocessor architectures utilizing 8-bit word lengths.

The (7,3) R-S code is a multiple error correcting code over GF(2^3) constructed to correct any two symbol errors per block. The basic code parameters are:

BLOCK LENGTH	$N = q - 1 = 7$
No. of Information Symbols:	$k = 3$
No. of Parity Check Symbols:	$N - k = 4$
Symbol Alphabet Size:	$q = N + 1 = 8$
Hamming Distance:	$d = N - k + 1 = 5$
Error Correction Capability: (symbols)	$t = \left[\frac{d-1}{2}\right]_{INT} = 2$

The elements of the extension field GF(2^3) can be represented as 3-tuples over the binary base field GF(2). All operations in this extension field are defined by a polynomial operation. The 3-tuples of the field can be generated by a 3rd degree irreducible primitive polynomial p(x) with coefficients over GF(2). Choosing p(x) = $x^3 + x + 1$ as such a polynomial, the $q = 2^3 = 8$ binary 3-tuple field elements are:

Field Element		Field Element Modulo - p(x)		Binary 3-Tuple
0	=		=	0 0 0
α^0	=	1	=	0 0 1
α^1	=	α^1	=	0 1 0
α^2	=	α^2	=	1 0 0
α^3	=	$\alpha + 1$	=	0 1 1
α^4	=	$\alpha^2 + \alpha$	=	1 1 0
α^5	=	$\alpha^2 + \alpha + 1$	=	1 1 1
α^6	=	$\alpha^2 + 1$	=	1 0 1

The generator polynomial g(x) from (5) becomes:

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) \qquad (8)$$

Since $- \alpha^i = + \alpha^i$, (i.e., subtraction and addition are equivalent in modulo-2 operation), equation (8) becomes:

$$g(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^3)(x + \alpha^4)$$

$$= x^4 + \alpha^3 x^3 + \alpha^0 x^2 + \alpha^1 x + \alpha^3 \qquad (9)$$

A1.3 Encoding

Recalling equation (1) for cyclic codeword generation we can construct a codeword for any message polynomial w(x). There are q^k unique messages represented by k-1 degree message polynomials of the form: $w(x) = w_{k-1} X^{k-1} + w_{k-2} X^{k-2} + . . . + wX + w_0$.

Example

The (7,3) Reed-Solomon code has a generator polynomial:

$$g(x) = X^4 + \alpha^3 X^3 + \alpha^0 X^2 + \alpha^1 X + \alpha^3$$

encoding the message $(\alpha^1\ \alpha^6\ \alpha^5)$, from (1):

$$\begin{aligned} c(x) &= w(x)\ g(x) \qquad (1) \\ &= (\alpha^1 X^2 + \alpha^6 X + \alpha^5)(X^4 + \alpha^3 X^3 + \alpha^o X^2 + \alpha^1 X + \alpha^3) \\ &= \alpha^1 X^6 + \alpha^3 X^5 + \alpha^o X^4 + \alpha^3 X^3 + \alpha^o X + \alpha^1 \end{aligned}$$

therefore the encoded codeword is:

$$(\alpha^1\ \alpha^3\ \alpha^o\ \alpha^3\ \alpha^o\ \alpha^1)$$

Note the X^{k-1} coefficient of the message polynomial is always equal to the X^{N-1} coefficient of the codeword, α^1 in this example. This is so because g(x) is a monic polynomial. However, none of the other message polynomial coefficients directly appear in the encoded codeword. Therefore, encoding using equation (1) results in the generation of Non-systematic codewords, i.e., codewords in which the message is not immediately discernable. To recover the message, given a received (possibly corrected) non-systematic codeword, another operation must be performed. Obviously, one such possible operation is to simply divide c(x) by g(x) to recover w(x).

Ideally, it would be better (quicker) if the message could be directly extracted from the received codeword without further manipulation.

This is possible via the generation of Systematic codewords. These codewords are generated so that the message sequence is itself part of the codeword. Systematic codeword generation represents a tradeoff between an increase in encoding complexity and the ease of message extraction given a received (corrected) codeword. Consider the following codeword formations.

Given a message polynomial,

$$w(x) = w_{k-1}\, X^{k-1} + w_{k-2}\, X^{k-2} + \ldots\ldots + w_1 X + w_o$$

multiply by X^{N-k},

$$X^{N-k}\, w(X) = w_{k-1}X^{N-1} + w_{k-2}X^{N-2} + \ldots\ldots + w_1 X^{N-k+1} + w_o X^{N-k}$$

divide by g(x) and express the result as a quotient q(x) and remainder r(x),

$$\underbrace{X^{N-k}\, w(x)}_{\deg\, \leq\, N-1} = \underbrace{q(x)g(x)}_{\deg\, \leq\, N-1} + \underbrace{r(x)}_{\deg\, \leq\, N-k-1}$$

or

$$r(x) + X^{N-k} w(x) = q(x)g(x) = c(x) = (\text{the desired systematic codeword})$$

$$= \underbrace{r_o + r_1 X + r_2 X^2 + \ldots\ldots + r_{N-k-1}\, X^{N-k-1}}_{\text{PARITY CHECK POLYNOMIAL}} +$$

$$\underbrace{w_o\, X^{N-k} + w_1 X^{N-k+1} + \ldots\ldots + w_{k-1}\, X^{N-1}}_{\text{MESSAGE POLYNOMIAL}} \qquad (10)$$

__EXAMPLE__:

For the (7,3) R-S code, consider the message $(\alpha^1\ \alpha^6\ \alpha^5)$,

$$w(x) = \alpha^1 X^2 + \alpha^6 X + \alpha^5$$

multiplying by X^{N-k},

$$X^4\ w(x) = \alpha^1 X^6 + \alpha^6 X^5 + \alpha^5 X^4$$

dividing by g(x), letting q(x) be the quotient and r(x) the remainder,

$$X^4\ w(x) \div g(x) = \underbrace{\alpha^1 X^2 + \alpha^3 X}_{q(x)} \quad \underbrace{(\alpha^5 X^3 + \alpha^6 X)}_{r(x)}$$

from (10)

$$c(x) = \underbrace{\alpha^1 X^6 + \alpha^6 X^5 + \alpha^5 X^4}_{w(x)} \quad \underbrace{(\alpha^5 X^3 + \alpha^6 X)}_{r(x)}$$

The systematic codeword is then $(\underbrace{\alpha^1\ \alpha^6\ \alpha^5}_{\text{msg}}\ \ \underbrace{\alpha^5\ 0\ \alpha^6\ 0}_{\text{parity-check}})$

For the short (7,3) R-S code used in the examples, there are only 512 possible codewords. Therefore, a simple table-look-up encoding procedure could be implemented by constructing a table of valid codewords and addressing the correct codeword in the table via a pointer determined by the message to be encoded. With this method of encoding it is clear that either a systematic or non-systematic approach is viable. One advantage of the systematic approach, however, is the immediate recognition of a correct encode operation. For decoding, however, systematic codewords clearly save the additional step of message extraction from the received (corrected codewords).

From the above analysis it may be concluded that for <u>short</u> block codes a table-look-up encoding procedure is applicable to systematic or non-systematic codeword generation. For <u>long</u> block codes the trade-offs necessary to effect such a solution involve table storage versus polynomial encoding time via equation (1) for non-systematic or (10) for systematic codewords.

A1.4 Decoding

Equation (1) or (10) shows that a valid codeword is divisable by its generator polynomial with no remainder. Given a received codeword polynomial r(x), it is possible to divide r(x) and interpret any non-zero remainder as evidence of transmission errors. Such a procedure determines what is known as the error syndrome.

$$\begin{aligned} r(x) &= c(x) + e(x) \\ &= w(x)\, g(x) + e(x) \end{aligned} \qquad (11)$$

$$\begin{aligned} R(\alpha^i) &= w(\alpha^i)\, g(\alpha^i) + e(\alpha^i) \\ &= e(\alpha^i) \triangleq S_j \ ; \ g(\alpha^i) = 0 \end{aligned}$$

Equation (11) illustrates that the syndrome components S_j contain information relative to the error pattern e(x).

If an array is formed with all possible codewords (including the all zero codeword) as the first row and all the correctable error patterns listed as the first column, ordered for increasing weight, (i.e., the number of non-zero components), the remainder of the array can be filled in by adding each codeword to every error pattern, making sure no N-tuple appears more than once. An array so constructed is called a Standard Array and each row is called a Coset with the error patterns heading the rows termed Coset Leaders. Equation (11) indicates that every error pattern has a unique syndrome associated with it. Therefore decoding a received codeword in the simplest sense requires (1) calculating the syndrome from the received (corrupted) codeword, (2) associating the calculated syndrome with its unique error pattern, and (3) knowing the corrupting error pattern, correcting the received codeword.

Although the decoding procedure as stated in the previous paragraph may sound simple enough, actual decoding algorithms to implement this procedure can be quite complex or lengthy. In the development of coding theory there have been many decoding techniques proposed. Using the standard array itself Coset Decomposition (Table Decoding)[7] and Step-by-Step decoding[5] have been proposed. For certain orthogonalizable codes, Threshold Decoding[8] or Majority Logic[5] decoding have proven effective.

Error trapping[7] and Permutation Decoding[5] have also been demonstrated as successful decoding strategies for some classes of cyclic codes.

Although all of the aforementioned decoding techniques are effective under varying code constraints, we have chosen to look at two vastly different decoding strategies. One approach, Maximum-Likelihood Decoding, is trivially simple in nature. The other, Algebraic Decoding, is a powerful but complex implementation of the basic decoding strategy outlined earlier.

A1.4.1 Algebraic Decoding

One of the truly revolutionary decoding algorithms to arise in the study of error coding is a decoding scheme based on the constructive properties of abstract algebra. By associating the digits of certain classes of linear cyclic codes, most notably the BCH codes, with elements of the finite field over which that code is defined, it is possible to define an "Error Locator Polynomial," whose roots reveal the location of the digits which are in error.

This algebraic BCH decoding algorithm[9] involves four distinct steps:

1. Calculation of the error Syndrome.
2. Determination of the coefficients of the Error Locator Polynomial.
3. Discovery of the error locations of the received (corrupted) codeword, i.e., the roots of the Error Locator Polynomial, and,
4. Calculation of the error values at the known error locations and correction of the received (corrupted) codeword.

The BCH decoding algorithm is an extremely powerful decoding technique and is the basis of much current decoding effort for long (multi-error correcting) block codes. The degree of computational complexity involved, however, is much greater than that easily handled by our objective single NMOS microprocessor and therefore a simpler algorithm, suited to short codes and simple computation, is sought.

A1.4.2 Minimum Distance Decoding (MDD)

The Distance between two codewords is defined as the number of symbols by which they differ. The Hamming Weight of a codeword is equal to the number of non-zero symbols in the codeword. The Minimum Distance of a code is the least distance between any two codewords in the code set. Since all zeros is a valid codeword, it can be stated that the minimum distance of a code is equivalent to the minimum weight of any non-zero codeword in the set. By employing these definitions a simple proof of the viability of MDD can be constructed.

Consider the following equivalencies:

R = a received data sequence (i.e., a codeword with errors)

C_i = the i^{th} codeword; the codeword transmitted

C_j = any j^{th} codeword $(j \neq i)$ not transmitted

E_j = $R-C_j$; the j^{th} error pattern resulting from mathematically combining the received data sequence and any valid j^{th} codeword except the one transmitted

E_i = $R-C_i$; the actual error pattern affecting the transmitted codeword

Denote the Hamming weight of E_j as $||E_j||$, then since

$$E_i = R-C_i$$

and

$$E_j = R-C_j$$

it follows that

$$E_j = E_i - C_j + C_i$$

$$\text{and} \quad ||E_j|| = ||E_i + (C_i - C_j)||. \qquad (12)$$

For proper (correct) decoding the weight of the actual error pattern must be less than or equal to the symbol error correction capability (t) of the code. Mathematically, the statement is

$$||E_i|| \leq t.$$

Because the code is a linear group code the combination of two codewords is simply another codeword and that codeword must have weight at least equal to the minimum distance (d) of the code or

$$||C_i - C_j|| \geq d$$

Analyzing the right hand side of equation (12), therefore, it can be seen that some error pattern E_i will contain the exact necessary error symbols in correct symbol positions to produce (reduce to zero) t more zero symbols in the combination pattern $E_i + C_i - C_j$ than in $C_i - C_j$ alone. In other words, the weight of $E_i + C_i - C_j$ may be as low as d - t. Since this represents the worst (minimum weight) case, the weight may be greater and equation (12) becomes,

$$||E_j|| = ||E_i + (C_i - C_j)|| \geq d - t \qquad (13)$$

Ideally, the weight of the error pattern E_j resulting from combining the received data sequence R with any codeword C_j not transmitted will be greater than the weight of the actual error pattern E_i or,

$$||E_j|| > ||E_i||$$

A sufficient condition for this to occur is,

$$d - t \leq ||E_j|| > ||E_i|| \leq t$$

so

$$d - t > t$$
$$d > 2t$$
$$d - 1 \geq 2t$$
$$\frac{d - 1}{2} \geq t \qquad (14)$$

Equations (13) and (14) show that if a transmitted codeword is corrupted by an error pattern with t or fewer symbol errors, that codeword will be decoded correctly by MDD. Alternatively, the distance (i.e., the weight of the error pattern) between a received (corrupted) codeword and any improperly decoded codeword (i.e., a codeword not transmitted) is at least *d - t if the* received (corrupted) codeword is distance (d - 1)/2 or less from the correct (transmitted) codeword.

An ERASURE is defined as a symbol error whose location is known. Merely knowing that a certain received symbol is in error without knowing the value of the error is helpful additional information in the decoding process.

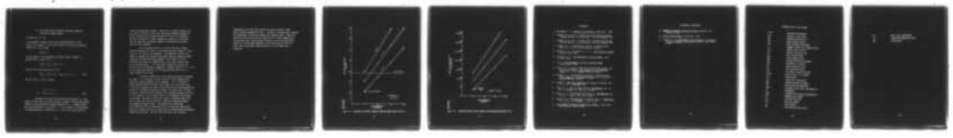
The preceeding MDD proof can be easily modified to include errors and erasures as follows: Let:

$\tilde{C}_i$ = the ith codeword; the codeword transmitted with s erased symbols deleted

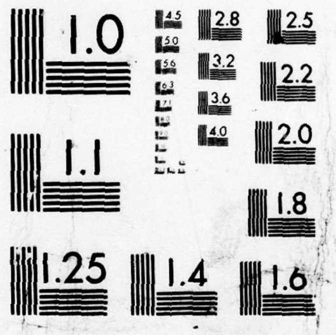
$\tilde{C}_j$ = any jth codeword ($j \neq i$) not transmitted with s erased symbols deleted

AD-A072 982 MITRE CORP BEDFORD MASS F/G 9/2
ERROR CORRECTION CODING WITH NMOS MICROPROCESSORS: CONCEPTS. VO--ETC(U)
MAY 79 E N SKOOG F19628-78-C-0001
UNCLASSIFIED MTR-3618-VOL-1 ESD-TR-79-125-VOL-1 NL

2 OF 2
AD
A072982



END
DATE
FILMED
9 -79
DDC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

$\tilde{R}$ = the received data sequence (corrupted codeword) including s erased symbols

as before $\tilde{E}_j = \tilde{R} - \tilde{C}_j$

If the number of symbol errors (not counting erasures) in the actual error pattern is less than or equal to the error-correction capability t of the code,

$$||\tilde{E}_i|| \leq t$$

and the weight of any codeword with erased symbols deleted is greater than or equal to d - s,

$$||\tilde{C}_k|| = ||\tilde{C}_i - \tilde{C}_j|| \geq d - s.$$

Similarly with the derivation of (13):

$$||\tilde{E}_j|| = ||\tilde{E}_i + (\tilde{C}_i - \tilde{C}_j)|| \geq d - s - t, \qquad (15)$$

and for $||\tilde{E}_j|| > ||\tilde{E}_i||$, we have

$$d - s - t > t$$

$$\text{or} \quad 2t + s \leq d - 1. \qquad (16)$$

Equations (15) and (16) show that the distance (i.e., the weight of the error and erasure pattern) between a received (corrupted) codeword and any improperly decoded codeword (i.e., a codeword which was not transmitted) is at least d - s - t or greater if the received corrupted codeword is distance (d - s - 1)/2 or less away from the correct (transmitted) codeword. Note that if neither equation (14) nor (16) are satisfied, the errors and erasure handling capability of the code has been exceeded and MDD will result in a decoding decision including the closest codeword or possibly an indication that the received (corrupted) codeword is equidistant from more than one valid codeword.

From the above proofs it is quite clear that minimum distance decoding will result in correctly decoding codewords if the limits of the code are not exceeded. The operations involved are simply to compare the received (corrupted) codeword symbol-by-symbol (ignoring the declared erasure symbols) with a table of valid codewords and choose the codeword that is the least distance away as the decoded choice. It is not always necessary to search the entire table, since anytime a codeword is located distance (d - s - 1)/2 or less away, equations (15) and (16) guarantee that there are no other codewords any closer.

In order to employ this trivial MDD algorithm for decoding R-S codes, it is necessary to have a table of valid codewords with which the received codeword can be compared. This is certainly a viable decoding method for short block codes but as the block size of the code increases the table size and search time grow exponentially. Figure A-1 indicates the required storage (in 8-bit bytes) for short R-S codes of differing block length N and information symbol content k. For our example (7,3) R-S code, a table of 512 codewords (1344 bytes) is required. Note that as the block length N increases, (shown only for R-S codes over binary extension fields), the required storage size becomes extremely large. The dashed line at 65K bytes illustrates the limit of addressability of current 8-bit MOS microprocessors without resorting to some form of extended memory management addressing techniques. Not only does the table size increase exponentially, but the time required to search through a large table increases exponentially also. Assuming a microprocessor requires 30 microseconds to fetch and compare one codeword symbol, Figure A-2 shows the table search time for one-half of all table codewords for varying code sizes. From storage and search time constraints alone, clearly our trivial MDD algorithm is only applicable to short R-S codes, given the present state-of-the-art of 8 bit NMOS microprocessors.

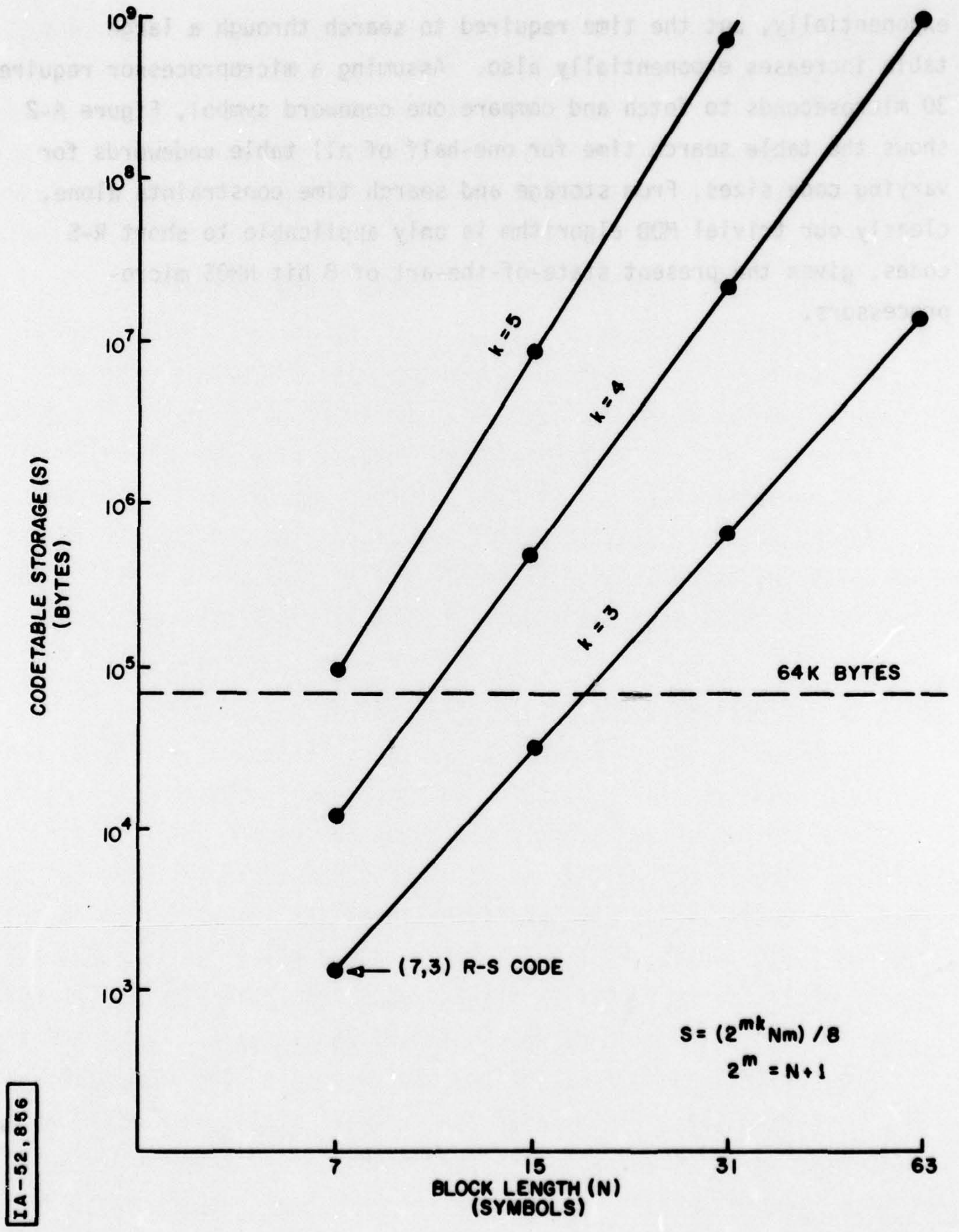


Figure A-1: CODETABLE STORAGE vs BINARY REED-SOLOMON CODE SIZE (N,k)

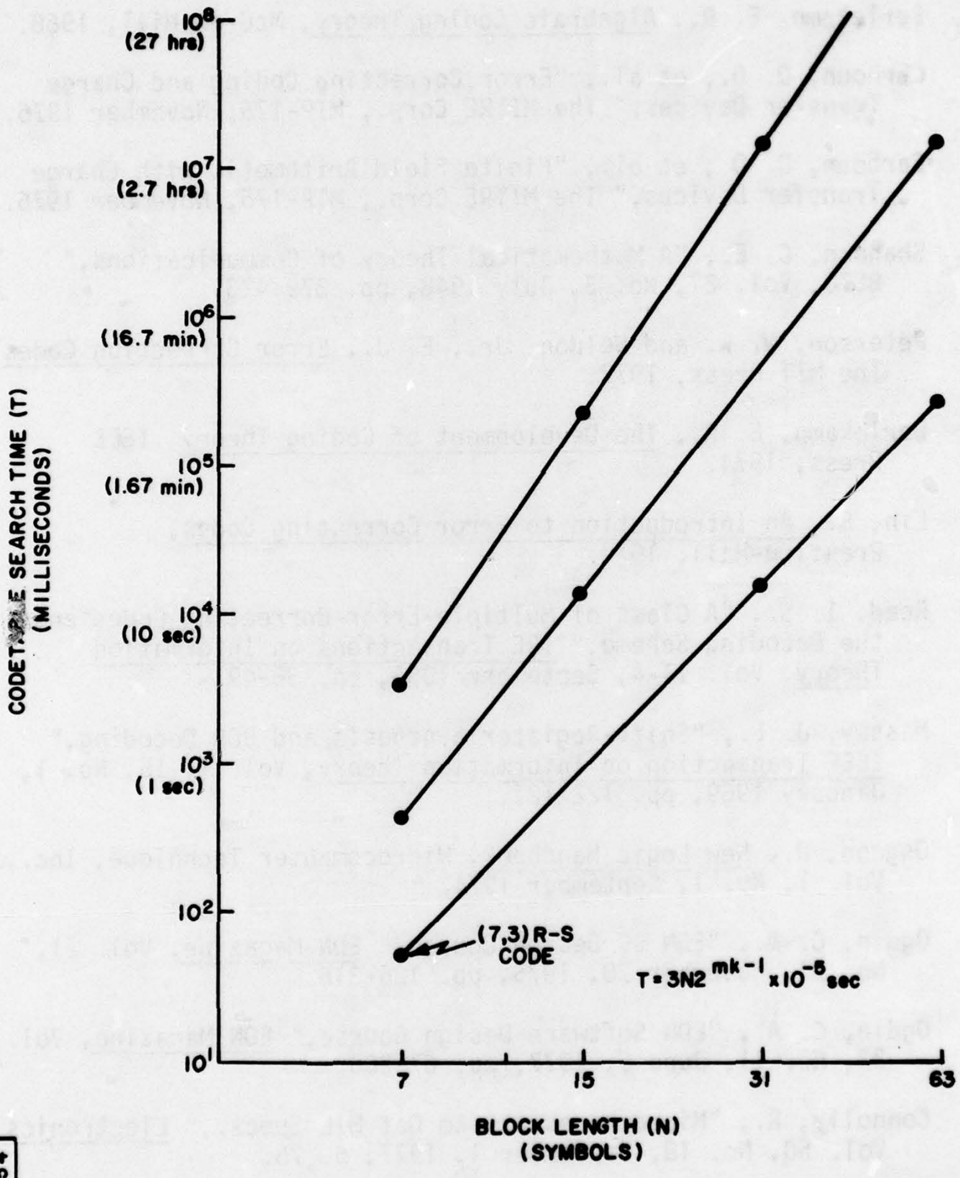


Figure A-2 : CODETABLE SEARCH TIME vs BINARY REED-SOLOMON CODE SIZE (N,k)

REFERENCES

1. Berlekamp, E. R., Algebraic Coding Theory, McGraw-Hill, 1968.

2. Carhoun, D. O., et al., "Error Correcting Coding and Charge Transfer Devices," The MITRE Corp., MTP-176, November 1976.

3. Carhoun, D. O., et al., "Finite Field Arithmetic with Charge Transfer Devices," The MITRE Corp., MTP-175, November 1976.

4. Shannon, C. E., "A Mathematical Theory of Communications," BSTJ, Vol. 27, No. 3, July 1948, pp. 379-423.

5. Peterson, W. W. and Weldon, Jr., E. J., Error Correction Codes, The MIT Press, 1972.

6. Berlekamp, E. R., The Development of Coding Theory, IEEE Press, 1974.

7. Lin, S., An Introduction to Error Correcting Codes, Prentice-Hall, 1970.

8. Reed, I. S., "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," IRE Transactions on Information Theory, Vol. IT-4, September 1954, pp. 38-49.

9. Massey, J. L., "Shift-Register Synthesis and BCH Decoding," IEEE Transaction on Information Theory, Vol. IT-15, No. 1, January 1969, pp. 122-127.

10. Osgood, R., New Logic Handbook, Microcomputer Technique, Inc., Vol. 1, No. 1, September 1974.

11. Ogdin, C. A., "EDN μC Design Course," EDN Magazine, Vol. 21, No. 21, November 20, 1975, pp. 126-316.

12. Ogdin, C. A., "EDN Software Design Course," EDN Magazine, Vol. 22, No. 11, June 5, 1977, pp. 67-200.

13. Connolly, R., "Microprocessors to Get MIL Specs.," Electronics, Vol. 50, No. 18, September 1, 1977, p. 76.

14. Intel 8080 Microcomputer Systems User's Manual, Intel Corp., July 1975, pp. 5-13 to 5-19.

REFERENCES (CONCLUDED)

15. M6800 Microcomputer System Design Data, Motorola, Inc., 1976, pp. 21-38.

16. Zilog Z-80 CPU Manual, Zilog Corp., 1976.

17. Osborne, A., An Introduction to Microcomputers, Volume II, Some Real Products, Adam Osborne and Ass., Inc., 1976, p. xxv.

ACRONYMS USED IN THIS REPORT

A/D/A	ANALOG/DIGITAL/ANALOG
ALU	ARITHMETIC LOGIC UNIT
AMD	ADVANCED MICRO DEVICES
BCD	BINARY CODED DECIMAL
BCH	BOSE-CHAUDHURI-HOCQUENHEM
C^3	COMMAND, CONTROL AND COMMUNICATIONS
CCD	CHARGE COUPLED DEVICE
CPU	CENTRAL PROCESSING UNIT
DIP	DUAL-INLINE-PACKAGE
DMA	DIRECT MEMORY ACCESS
DME	DISTANCE MEASURING EQUIPMENT
DOD	DEPARTMENT OF DEFENSE
DTL	DIODE-TRANSISTOR LOGIC
ECL	EMITTER-COUPLED LOGIC
ESD	ELECTRONIC SYSTEMS DIVISION
GF(q)	GALOIS FIELD OF q ELEMENTS
I/O	INPUT/OUTPUT
LSI	LARGE SCALE INTEGRATION
MDD	MINIMUM DISTANCE DECODING
MOS	METAL-OXIDE-SEMICONDUCTOR
NMOS	N-CHANNEL METAL-OXIDE SEMICONDUCTOR
nS	NANOSECOND
PMOS	P-CHANNEL METAL-OXIDE-SEMICONDUCTOR
RAM	RANDOM ACCESS MEMORY
RC	RESISTOR-CAPACITOR
ROM	READ ONLY MEMORY
R-S	REED-SOLOMON
RTL	RESISTOR-TRANSISTOR LOGIC

SSI	SMALL SCALE INTEGRATION
TTL	TRANSISTOR-TRANSISTOR LOGIC
XOR	EXCLUSIVE-OR